REPRESENTATION LEARNING FOR WEB INTELLIGENCE

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Yixuan Li

December 2017

REPRESENTATION LEARNING FOR

WEB INTELLIGENCE

Yixuan Li, Ph.D.

Cornell University 2017

The excitement of web-driven machine learning research comes at two different granularity levels: on one hand, it enables us to model the complex interactions among web users and web components in the abstract; and on the other hand, it provides us with an automated means for comprehension of concrete web contents such as texts, images and audios. Through the lens of machine learning and machine vision, this thesis focuses on approaches of learning representations of web-scale data in perspectives of *web user interaction* and *web content comprehension*.

This thesis work contributes to the field of machine learning in both algorithmic foundations and practical advances. The first part focuses on classification and prediction problems on the web when the interactions of individuals are modeled by graphs. We will leverage graphs and machine learning models as tools to effectively represent how users' interactions are related to each other, and to identify interesting patterns of behavior of web users. The second part of the thesis focuses on methods for understanding and comprehending visual contents on the web. In particular, we turn to deep learning – a subfield of machine learning that has recently been placed at the core of many web-driven tasks such as image classification and understanding. Despite its success, there has been lacking understanding of the representations learned by deep neural networks. This thesis sheds light on how we can be better interpret the inner

representations of deep neural networks, and also leads to new techniques of how we can facilitate learning better representations.

To my husband Gabriel Culbertson. For everything.

ACKNOWLEDGEMENTS

First and foremost I wish to thank my advisor, Professor John Hopcroft, who has been supportive to my research pursuit throughout my PhD study. John has supported me not only financially by providing a research assistantship over the past years, but also academically by encouraging me to explore frontier research questions in computer science. He lent me the great research vision, and taught me to define career path through constantly self-questioning *where do I see myself in the next 5-10 years*.

My thesis committee guided me through all these years. Thank you to Professor Thorsten Joachims and Professor Kilian Q. Weinberger for being on my committee. Over the past years, Kilian has been more than just a minor advisor to me – he kindly participated in this research by giving generously of his time and mentorship and providing me access to his wonderful students.

My sincere thanks also goes to Jason Yosinski and Jeff Clune, who provided me with a collaboration opportunity, and who gave access to the research facilities. Without their precious support it would not be possible to accomplish the research leap onto deep learning.

A massive thank you to my friends and colleagues from Kilian's research group, for their willingness to discuss research ideas, and for making it such a great hub to work: Gao Huang, Shuang Li, Yu Sun, Matt Kusner, Geoff Pleiss, Chuan Guo, Felix Wu, Zhuang Liu and Jake Gardner.

My appreciation extends to collaborators and mentors from internships at Google (Research) and GrokStyle: Kavita Bala, Sean Bell, Vidhya Navalpakkam, Dmity Lagun, Pingmei Xu, Oscar Martinez, Xing Chen and Yi Li. Thank those who attended my talks and posters, and those who emailed me about my work, with exciting ideas, questions, and discussions about research, in particular

Kyle Kloster.

Dozens of people have helped and taught me immensely at Cornell over the years. I would like to thank all my friends for making life in Ithaca so enjoyable; and colleagues from Computer Science, Information Science and Statistics (CIS) for cultivating the friendly and stimulating research atmosphere. My thesis research has also greatly benefited from the Artificial Intelligence Seminar series, Machine Learning Discussion Group organized by fellow PhD students, and reading groups at many facets.

I would like to thank wholeheartedly to my family: my parents Zhijun Li and Mingfang An, my parents-in-law Hal and Rhonda Culbertson, and to my brother Yichen Li and sister-in-law Naomi Culbertson for supporting me with love in life. And my deepest thank goes to my husband Gabriel Culbertson, for being willing to join me for the journey and provide constant accompany and support. I would never make this far without him.

Finally, thank you to School of Electrical and Computer Engineering for supporting my graduate work and for embodying the spirit of Cornell: an institution where any person can find instruction in any study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The excitement of web-driven machine learning research comes at two different granularity levels: on one hand, it enables us to model the complex interactions among web users and web components in the abstract; and on the other hand, it provides us with an automated means for comprehension of concrete web contents such as texts, images and audios. Through the lens of machine learning and machine vision, this thesis focuses on approaches of learning representations of web-scale data in perspectives of *web user interaction* and *web content comprehension*.

The first part focuses on classification and prediction problems on the web when the interactions of individuals are modeled by graphs. We will leverage graphs and machine learning models as tools to effectively represent how users' interactions are related to each other, and to identify interesting patterns of behavior of web users. The second part of the thesis focuses on methods for automatically understanding and comprehending visual contents on the web. In particular, we will turn to deep learning – a subfield of machine learning that has recently been placed at the core of many web-driven tasks such as image classification and understanding. The representation power of neural networks is enabled through learning a hierarchy of concepts, with each concept built out of simpler ones. The ever-growing power in model expressiveness has exposed new research challenges and directions to the field. In the second part I will

highlight a few contributions to deep learning research.

1.1 Part I: Machine Learning for Modeling Web User Interactions

Graphs are powerful tools for understanding and modeling complex interactions on the web. In many cases, web data can be effectively modeled using a relational graph, where users are nodes, and interactions between users are edges. A subset of nodes sharing certain common characteristics can be viewed as a community. For example, in social networks, a node (user) might belong to a work community, a community of friends, and a community of individuals that share the same hobby.

The first part of the thesis is comprised of three chapters. Firstly, we introduce a principled framework for scalable graph clustering. The proposed semi-supervised learning method, LEMON, can effectively learn representations of local graph structure embedding. By leveraging the latent representations, we show promise of finding small clusters in large graphs in a localized manner without relying on the global graph structure (Chapter 1). I will then demonstrate a Google-scale deployment of such algorithm, which significantly reduces the computational cost compared to state-of-the-art approaches (Chapter 2). Furthermore, going beyond static network structure, we show how machine learning approaches can effectively model the dynamics of online groups evolving on the web (Chapter 3).

Chapter 1 considers the general semi-supervised learning problem of labeling nodes around a given seed node or set of seed nodes of interest. That is, the al-

gorithm is given as input a node (or nodes) in the graph, and the goal is to find a cluster (community) of which it is a member. The problem of clustering nodes into smaller groups (communities) is classical in machine learning, and has been an important graph primitive with a wealth of applications and research activity. Conventional approaches relying on the global graph structure is no longer feasible as graph scales to contain billions of nodes. The rapid growth of web-scale data calls for solutions that are both computationally efficient and scalable. To this end, we introduce a principled framework for scalable graph learning using localized probabilistic diffusion approach. Specifically, we propose a novel approach named LEMON (*Local Expansion via Minimum One Norm*). LEMON can scale effectively to extremely large relational graphs since the computations involve only partial of the graph. The general framework operates by first conducting a local diffusion, which propagates probability values from the labeled nodes (seeds) to the remaining unlabeled ones. The algorithm then returns as the detected community the set of nodes that have largest probability mass. We demonstrate that LEMON can achieve high detection accuracy with significantly reduced computation cost – outperforming state-of-the-art proposals by a large margin. We release our code and database publicly to support future research, available online at `https://github.com/yixuanli/lemon`.

Chapter 2 puts the principles of local graph clustering into practice, and applies the ideas to large-scale deployment at Google. We present *Local Expansion at Scale* (LEAS), a large-scale implementation of local spectral clustering algorithm developed in Chapter 1. We focus on solving the practical problem of detecting fake/abusive accounts on the social site of YouTube. Generally speaking, any web user activity that does not reflect user's genuine interest can be viewed as fake social engagement. We show how those fake engagement ac-

tivities on YouTube can be tracked over time by analyzing the temporal graph based on the engagement behavior pattern between users and YouTube videos. The engagement graph allows us to detect orchestrated actions by sets of users which have a very low likelihood of happening spontaneously or organically. Such behavior of groups of users acting together on the same web content at around the same time is also known as lockstep behavior.

To detect the lockstep behavior, we make use of existing known abusive accounts as seeds. Specifically, LEAS searches for nodes with similar pattern of behavior as the given seeds. We offer a fast, scalable MapReduce deployment adapted from the localized spectral clustering algorithm. We demonstrate the effectiveness of our deployment at Google by achieving accuracy of 98%. Comparing with the state-of-the-art approach, LEAS achieves 10 times faster running time on average. Leas is now actively in use at Google, searching for daily deceptive practices on YouTube spanning over a billion users. Our approach is generally applicable, and can be extended to many other real-world settings including Twitter followers, Amazon product reviews and Facebook Likes etc.

Chapter 3 goes beyond discovering communities in static relational graphs, and investigates also the dynamics of interactions behind the online groups (communities) – with the goal of understanding the processes by which groups come together, grow new members, and evolve over time.

In collaboration with Tencent Corporation, we analyzed the real-world data from WeChat group messaging platform – the largest social messaging service in China, with more than 600 million monthly active users. WeChat, similar to WhatsApp, allows users to send and receive multimedia messages in real-time via Internet. One compelling feature in WeChat is group chat. A chat group on

WeChat can be analogy to a community, where one can chat with several users at a time. There are two ways in which a user can involve in a chat group: one can either initiate a new chat group, or get invited by an existing member of the group. Groups play a very important role in WeChat. Our statistics show that group chat constitutes 25% of the total message traffic on WeChat. More than 2 million new groups are created every day. To our knowledge, this is by far the first large-scale rigorous study on modeling WeChat group dynamics.

We model the growth and evolution of messaging groups at two levels: group level and individual level. At the group-level, we predict whether a social group will grow and persist in the long run, using signals from the structural and behavioral patterns exhibited by the group at its early stage. At the individual-level, we model the process by which groups gain new members. Specifically, can we predict which users in the group are more likely to be active and invite new users, and to whom will they send invitations? Making sense of such questions requires fine-grained inspection into users' historical engagement behavior as well as the local social network structure that users are embedded in. To this end, we develop a membership cascade process model in which we consider features of both inviter – a group member who sends invitation to friend(s), and invitee – the individual in the inviter's ego networks who gets invited to the group chat. Our inviter prediction model using all features generally achieves prediction accuracy as high as 95.31%, and invitee prediction model reaches accuracy of 98.66%.

1.2 Part II: Machine Vision for Understanding Web Contents

Apart from modeling the interactions among web entities, an integral part of the thesis work further studies approaches for analyzing and comprehending web contents such as visual images. Many visual understanding tasks (e.g., tell apart a lion and a jaguar, read a sign, or recognize a human's face) seem easy to human brains, yet they've been hard problems to solve with computers for decades.

In the past few years, the field of machine learning has made tremendous progress on addressing these difficult problems. In particular, deep learning, a branch of machine learning algorithms, has recently been working really well - matching or exceeding human performance in some visual understanding tasks. A deep neural network (DNN) usually consists of multiple hidden layers of neurons between the input and output layers. DNNs are effective in modeling complex nonlinear relationships. The expressiveness of neural networks is enabled through learning a hierarchy of concepts, with each concept built out of simpler ones. Researchers have demonstrated steady progress in computer vision, each time achieving a new state-of-the-art result: LeNet [113], AlexNet [100], VGG-Net [161], ResNet [69] and DenseNet [76]. While successive models continue to show improvements, the ever-growing power in model expressiveness has exposed new research challenges and directions to the field.

First, deep neural networks should be treated more than just expressive black-box models, and call for a greater understanding and intuition for the computation performed in the middle. Researchers have only recently started investigating the features learned on their intermediate layers. Such research is

difficult because it requires making sense of non-linear computations performed by millions of parameters, but valuable because it increases our ability to understand learned representations and create improved versions of them. This part of my thesis focuses on develop mechanisms for the purpose of better understanding the inner workings of deep neural networks (Chapter 4). Built upon the understandings, we also explore approaches for improving the performance of deep neural networks through collaborative training of multiple neural networks, where we leverage both common features shared across models as well as model-specific features (Chapter 5). Lastly, the great model expressiveness comes at a non-trivial computational cost. As model complexity grows, even a single neural network can take days or weeks to train on GPUs, not to mention ensembles of neural networks. This becomes uneconomical for most researchers without access to industrial scale computational resources. We discuss mechanism to obtain cheap neural network ensembles with saving computational cost (Chapter 6).

Chapter 4 investigates the extent to which neural networks exhibit what we call *convergent learning*, which is when the representations learned by multiple nets converge to a set of features which are either individually similar between networks or where subsets of features span similar low-dimensional spaces. We propose a specific method of probing representations: training multiple networks (with the same architecture but different random weight initializations) and then comparing and contrasting their individual, learned representations at the level of neurons or groups of neurons. We use correlation statistics as a way of measuring how related the activations of one neuron are to another neuron, either within the network or between networks. To find equivalent or nearly-equivalent neurons across networks, we adopt the following three tech-

niques: a bipartite matching approach that makes one-to-one assignments between neurons, a sparse prediction approach that finds one-to-many mappings, and a spectral clustering approach that finds many-to-many mappings.

This initial investigation reveals a few previously unknown properties of neural networks, and we argue that future research into the question of convergent learning will yield many more. The insights described here include (1) that some features are learned reliably in multiple networks, yet other features are not consistently learned; (2) that units learn to span low-dimensional subspaces and, while these subspaces are common to multiple networks, the specific basis vectors learned are not; (3) that the representation codes show evidence of being a mix between a local code and slightly, but not fully, distributed codes across multiple units.

We further discovered that the matching of features is more common to the first several layers than higher levels. The first level of the network learns features of images such as Gabor filters, color and edge detectors, etc. but not features of the specific set of images. The deeper layers in the networks tends to converge to slightly different sets of features, increasing the difficulty of finding the matching features. We release our code and database publicly to support future research, available online at `https://github.com/yixuanli/convergent_learning`.

Chapter 5 builds on the insights gained from Chapter 4, and presents *Subspace Ensemble Networks* (SEN), a method that improves the generalization performance of small deep network ensembles through learning a shared feature subspace. In large part, the success of ensembles is due to their having significantly lower variance than any individual classifier. Unfortunately, the massive

training time of state-of-the-art deep networks largely prohibits their use in ensembles larger than size 5. This is in contrast to the 500-model ensemble used to win the Netflix prize competition.

SEN provides an elegant framework to trade-off the variance of individual deep networks with a carefully chosen model bias. We facilitate this by learning a *shared representation subspace*, that can be aligned across all classifiers of the ensemble, alongside network-specific *null spaces*. By enforcing shared classifier weights in the common space we bias the ensemble members towards a common classifier, which is trained on the internal representations of all networks. This fine-tuned bias-variance trade-off enables us to drastically reduce the ensemble sizes, in most datasets outperforming standard ensembles of 20 deep networks *with just 2*—resulting in improved accuracy and an order of magnitude reduction in training time. We release our code and database publicly to support future research, available online at `https://github.com/yixuanli/sen`.

Chapter 6 proposes an efficient method to obtain an ensemble of multiple neural networks with significant deduction in computation costs. The ensemble method – averaging the predictions of multiple machine learning models – is an effective method to improve prediction accuracy by approximating the expected classifier. Ensembles of deep networks have been shown to yield classifiers that even surpass human abilities. Some of the state-of-the-art competition-winning approaches for image classification have simply been ensembles of several deep learning architectures. However, training deep networks is computationally expensive and can last for days or even weeks even on high performance hardware with GPU acceleration. Training ensembles of them increases the cost lin-

early and quickly becomes prohibitive for most researchers without access to industrial scale computational resources. Although the training of deep net ensembles can be trivially parallelized, few have access to sufficient GPU servers that can be deployed in parallel for long durations.

To this end, we introduce an efficient ensemble method by letting a single neural network converge into several local minima along its optimization path and save the intermediate model parameters. The resulting technique, which we refer to as Snapshot Ensembling, is surprisingly simple, yet effective. We show in a series of experiments that our approach is compatible with diverse network architectures and learning tasks. It consistently yields significantly lower error rates than state-of-the-art single models at no additional training cost, and matches favorably with the results of (far more expensive) independently trained network ensembles. On CIFAR-10 and CIFAR-100 datasets, our approach has achieved a new record on image classification, with error rates of 3.4% and 17.4% respectively.

Part I

Machine Learning for Modeling

Web User Interactions

CHAPTER 2

SCALABLE GRAPH LEARNING: THE PRINCIPLES

This chapter is written in collaboration with David Bindel, John Hopcroft, Kyle Kloster and Kun He. The work was published in Proceedings of the 24th International Conference on World Wide Web in 2015. Extended journal version has been accepted for publication in the Transactions of Knowledge and Data Discovery (TKDD) in 2017.

2.1 Introduction

Analyzing the structure and extracting information from complex networks is an important research area. Significant research has been carried out in finding the structure of networks and identifying communities [56].

In early work, researchers assumed that communities were disjoint and had more internal connections than external connections. Both assumptions have been discarded since it is clear that in most networks a vertex belongs to more than one community. For instance, in social networks, one might belong to a work community, a community of friends, and a community of individuals that share the same hobby such as golf; in co-purchased networks, one item might belong to multiple categories. Also since we are dealing with networks with hundreds of millions of vertices, an individual in a community of size 100[1] will certainly have more links outside the community than inside. These key insights have motivated us to identify communities from a new perspective.

[1] A statistical study on social networks done by Leskovec *et al.* [119] has shown that real-world communities with high quality are quite small and usually consist of no more than 100 vertices.

Considerable researches on detecting communities have focused on the global structure. And these globally based detection algorithms usually run in time functional to the size of the entire graph, a major drawback in computational cost. Nowadays, we explore networks with billions of vertices to find communities of size a hundred. Thus, taking the entire graph into account might not serve as a practical solution in many situations. It is thus crucial to shift our attention from global structure to local structure in large networks, and develop new approaches that enable finding communities in time functional to the size of the community.

Quite recently, there has been a growing interest in finding communities by locally expanding an exemplary seed set in the community of interest [10][95][184][190]. This type of algorithm usually starts with a few members that are already known to be in the target community, and the goal is to uncover the remaining members in the community as the exemplary members. These known members are usually referred to as *seeds* in the literature, and the process of growing the seed set gradually into a larger set until the target community is revealed is called *seed set expansion*. The setting of seed set expansion can be widely applied to real world applications. For example, in web search, with a few known pages that share similar information, we could generate a larger group of web pages that contains the relevant contents with respect to a certain search query; in product networks, seed set expansion enables the automatical categorizing of products that are discovered to be in the same community as the labeled items.

The random walk technique has been extensively adopted as a subroutine for locally growing the seed set in the literature [10][84][95][149][155][184][190].

The dynamics of random walks are effective in finding a local community since they make non-uniform expansion decisions based on the structure revealed during the exploration of the neighborhood surrounding the seeds [10]. This implies that random walk based local expansion is able to trace the community members in a principled way that best resembles the natural process for forming the local community structure. Very recently, Abrahao *et al.* also experimentally verified that random walk produces communities that are most structurally similar to real-world communities amongst various algorithmic communities [6].

In this paper we propose a novel approach for finding overlapping communities called LEMON (Local Expansion via Minimum One Norm)[2] for finding overlapping communities in large networks. We systematically demonstrate that LEMON can achieve both high efficiency and effectiveness that significantly stands out amongst state-of-the-art proposals. Specifically, we consider the span of a few dimensions of vectors after the short random walk and use it as the approximate invariant subspace, which we refer as *local spectra*.

In contrast to the traditional spectral clustering methods, our local spectral method does not require the burdensome computation of a large number of singular vectors. In addition, as traditional spectral methods usually partition the vertices into disjoint communities, we make another fundamental change. Concretely, we mine the communities from the subspace by seeking a sparse approximate indicator vector in the span of the local spectral such that the seeds are in its support. In practice, this can be mathematically achieved by solving a ℓ^1-penalized linear programming problem.

[2]Our demo code is publicly available at: `https://github.com/yixuanli/lemon`.

Domain	Dataset	Vertices	Links	Average membership	Maximum membership	Community size mean
Product	Amazon	334,863	925,872	0.11	49	39
Collaboration	DBLP	317,080	1,049,866	0.22	11	251
Social	YouTube	1,134,890	2,987,624	0.05	41	79
Social	Orkut	3,072,441	117,185,083	9.56	504	83

Table 2.1: Statistics for the real networks.

We aim to develop a comprehensive understanding of the local spectral approach for identifying a community from a small seed set. Following the central idea of our approach, we seek to solve fundamentally important questions such as: what defines "good" communities and when do they emerge as we expand the seed set (Section 2.4.4)? How to find a small community in time functional to the size of the community rather than that of the entire graph (Section 2.4.3)? What defines "good" seeds and how many seeds could uniquely define a community (Section 2.5)? And given that networks are not all similar in nature, how the local expansion approach is suited for uncovering communities in different types of networks (Section 2.6.4)?

We thoroughly evaluate our approach using both synthetic and real-world datasets across different domains, and analyze the empirical variations when applying our method to inherently different networks in practice. We believe that the insights we gained from researching on these problems would provide valuable guidance for future investigation on this topic.

2.2 Related Work

A considerable amount of literature has been published on finding communities in large social and information networks. We highlight a few ideas that have

recently emerged in the literature to clarify how our method differs.

Globally based community finding algorithms. Various community detection algorithms have been developed in the past decade. And most of the algorithms fall into the category of global approach. One stream of global algorithms attempt to find communities by optimizing an objective function. For example, GCE [115] identifies maximal cliques as seed communities. It expands these cliques by greedily optimizing a local fitness function. OSLOM [109] is also based on the optimization of a fitness function, which expresses the statistical significance of clusters with respect to random fluctuations (i.e., the random graph generated by the configuration model [132] during community expansion). However, the communities identified by mathematical construction may structurally diverge from real communities as pointed in [6]. Another main stream of research adopts the label propagation approach [151], which defines rules that simulate the spread of labels of vertices in the network. The DEMON algorithm [42], for example, democratically lets each vertex vote for the communities it sees surrounding it in its limited view of the global system using a label propagation algorithm, and then merges the local communities into a global collection. Other approaches such as Link Community (LC) [8] partitions the graph by first building a hierarchical link dendrogram according to the link similarity and then cutting the dendrogram at some threshold to yield link communities.

Random walk based detection algorithms. As noted in the preceding section, among the divergent approaches, random walks tend to reveal communities that bear the closest resemblance to the ground truth communities in nature [6]. In the following, we briefly review some methods that have adopted the

random walk technique in finding communities. Speaking of methods that focus on the global structure, Pons *et al.* [149] proposed a hierarchical agglomerative algorithm, *WalkTrap*, that quantified the similarity between vertices using random walks and then partitioned the network into non-overlapping communities. Meil*ă et al.* [130] presented a clustering approach by viewing the pairwise similarities as edge flows in a random walk and studied the eigenvectors and values of the resulting transition matrix. A later successful algorithm, *Infomap*, proposed by by Rosvall & Bergstrom [155] enables uncovering hierarchical structures in networks by compressing a description of a random walker as a proxy for real flow on networks. Variants of this technique such as biased random walk [200] has also been employed in community finding.

Local expansion based approaches. To interpret the problem of community detection from a local perspective, our work shares the same spirit as the local expansion algorithms in [10], [95], [184] and [93]. Specifically, Andersen & Lang [10] adapted the theoretical results from [165] to expand a set into a community with locally minimal conductance based on lazy random walks. However, the lazy random walk endured a much slower mixing speed and it usually took more than 500 hundred steps to converge to a local structure compared with several steps of rapid mixing in a regular random walk. Featuring on the seeding strategies, Whang *et al.* [184] established several sophisticated methods for choosing the seed set, and then used similar PageRank scheme as that in [9] to expand the seeds until a community with optimal conductance is found. Nonetheless, the performance gained by adopting these intricate seeding methods was not significantly better than that by using random seeds. This implies that a better scheme of expanding the seeds is also needed aside from a good seeding strategy. A recent work by Kloumann & Kleinberg [95] provided a sys-

tematic understanding of variants of PageRank-based seed set expansion. They showed many insightful findings regarding the heuristics on seed set. However, the drawback of lacking a proper stop criterion has limited its functionality in practice. Even though a recently proposed heat kernel algorithm [93] advances PageRank by introducing a sophisticated diffusion method, the detection accuracy achieved by heat kernel approach is still much lower than that of LEMON, which we will show in Section 2.6.2.

Local spectra vs. global spectra. Spectral methods is one of the most widely used techniques for exploratory data analysis, with applications ranging from data clustering, image segmentation to community detection etc. Spectral clustering makes use of the first few singular vectors of the Laplacian matrix associated with a graph, which are inherently global quantities and may not be sensitive to very local information. For example, in the case when provided with domain knowledge about a target region in the graph, one might be interested in finding clusters *only* near the specified local region in a semi-supervised manner, which might not be otherwise well captured by a method using global eigenvectors. Therefore, in the semi-supervised setting, our pioneer work on local spectral clustering [121, 70][3] have substantial advantage over traditional spectral techniques, with the capability of prioritizing and learning more about a local region of the graph surrounding the seeds. Although the local spectral proposal in [127] incorporates the local information as an additional constraint based on the global spectral methods, the optimization program involves the entire eigenspace, which is less advantageous than using the partial invariant subspace constructed by the *Krylov subspace* in our approach.

[3]This manuscript is an extended version of an earlier conference publication [121].

2.3 Preliminaries

2.3.1 Problem Statement

Given a network $G = (\mathcal{V}, \mathcal{E})$ and a set of members $\mathcal{S}$ in the target community C, where $|C| \ll |V|$ and $|\mathcal{S}| \ll |C|$, we are interested in discovering the remaining members in C. Generally speaking, we focus on answering **how to accurately find a small community in time functional to the size of the community from a seed set?**

2.3.2 Symbols and Definitions

Table 2.2 summarizes a list of the different symbols we will use throughout the paper. In general, we use italic letters, e.g., n, μ, to denote scalars; lower boldface characters, e.g. $\mathbf{y}$, to denote vectors; uppercase boldface characters, e.g., $\mathbf{A}$, to denote matrices; and script characters, e.g., $\mathcal{C}$, to denote sets.

2.3.3 Datasets

Synthetic datasets

The LFR benchmark graphs [108] have been widely adopted for the purpose of evaluating the performance of community detection algorithms. LFR datasets are generated with built-in community structure that resembles the features found in most real-world networks with power-law degree distribution. It pro-

Symbol	Definiton and description
$\mathcal{S}$	Seed set
C	Detected community
C^*	Ground truth community
$G_{\mathcal{S}}$	Subgraph extracted from the neighborhood surrounding the seed set $\mathcal{S}$
N	Size of the subgraph $G_{\mathcal{S}}$
$\mathbf{A}_{\mathcal{S}}$	Adjacency matrix of subgraph $G_{\mathcal{S}}$
$\bar{\mathbf{A}}_{\mathcal{S}}$	Normalized adjacency matrix of subgraph $G_{\mathcal{S}}$
$\mathbf{D}_{\mathcal{S}}$	Diagonal degree matrix of subgraph $G_{\mathcal{S}}$
$\mathbf{L}_{\mathcal{S}}$	Laplacian matrix of subgraph $G_{\mathcal{S}}$
$\bar{\mathbf{L}}_{\mathcal{S}}$	Normalized Laplacian matrix of subgraph $G_{\mathcal{S}}$
$\mathbf{V}_{k,l}$	l-dimensional local spectral subspace with k-step random walks.
$\Phi(\mathcal{V})$	Conductance of the node set $\mathcal{V}$
$\lambda_i^{(\mathbf{H})}$	The i-th smallest eigenvalues of matrix $\mathbf{H}$
$\mathbf{y}$	Probability indicator vector, where larger value indicates a higher possibility being in the same community as the seeds

Table 2.2: Symbols and Definitons.

vides researchers with rich flexibility to control the network topology by tuning different parameters, including the graph size n, the average degree $\bar{k}$, the maximum degree k_{max}, the minimum and maximum community size $|C|_{min}$ and $|C|_{max}$, the mixing parameter μ, the overlapping membership om and the number of vertices with overlapping membership on. Among these parameters, the mixing parameter μ has the most significant impact on the network topology, which controls the fraction of links for each vertex that cross to a community with which the vertex is not associated. Usually, larger μ would result in lower detection accuracy.

Xie et al. [187] have performed a thorough performance comparison of different state-of-the-art overlapping community detection algorithms on LFR benchmark datasets. To make the performance evaluation of our algorithm consistent with that in [187], we adopt the same parameters in our paper. In total,

we generate two sets of networks with mixing parameter $\mu = 0.1$ and $\mu = 0.3$ respectively. We vary the parameter *om* from 2 to 8 for each μ and obtain a total of 14 networks. Table 2.3 lists the value of the parameters we have used for generating the LFR datasets.

Parameter	Description	Value
n	graph size	5000
μ	mixing parameter	$\{0.1, 0.3\}$
$\bar{k}$	average degree	10
k_{max}	maximum degree	50
$\|C\|_{min}$	minimum community size	20
$\|C\|_{max}$	maximum community size	100
τ_1	node degree distribution exp.	2
τ_2	community size distribution exp.	1
om	overlapping membership	$\{2, 3, ..., 8\}$
on	overlapping node	2500

Table 2.3: Parameters for the LFR datasets.

Real datasets

For the purpose of testing on real networks, we include four datasets with ground truth community membership from Stanford Network Analysis Project[4]. These datasets span various domains of network applications, including product networks (Amazon), collaboration networks (DBLP), and online social networks (YouTube and Orkut)[5]. Each of the networks can be viewed as an undirected, connected graph. The statistical information of the datasets is summarized in Table 2.1.

[4] `http://snap.stanford.edu`

[5] For all the four real datasets, we adopt the top 5000 communities that possess the highest quality according to [190].

2.3.4 Evaluation Metric

For the evaluation metric, we adopt F1 score to quantify the similarity between the algorithmic community C and the ground truth community C^*. The F1 score for each pair of (C, C^*) is defined by:

$$F_1(C, C^*) = \frac{2 \cdot Precision(C, C^*) \cdot Recall(C, C^*)}{Precision(C, C^*) + Recall(C, C^*)}, \tag{2.1}$$

where the precision and recall are defined as:

$$Precision(C, C^*) = \frac{|C \cap C^*|}{|C|}, \tag{2.2}$$

$$Recall(C, C^*) = \frac{|C \cap C^*|}{|C^*|}. \tag{2.3}$$

Throughout the paper, unless otherwise pointed out, the experimental results on synthetic data for each instance are given by the statistical mean and standard deviation based on 24 test cases[6]; and the experimental results on real datasets for each instance are based on 120 test cases. All the ground truth communities for testing are randomly chosen. The randomness of batch tests can guarantee the elimination of statistical bias in our tests.

2.4 Local Spectral Clustering

2.4.1 Algorithm Overview

Spectral clustering makes use of a small number of singular vectors proportional to the number of communities in the network. If a graph has thousands

[6]Each local expansion process from a seed set can be viewed as a test case.

of small communities, it is impractical to calculate a number of singular vectors greater than the number of communities. We are experimenting with a fundamentally new technique, which does not require the burdensome computation of a large number of singular vectors. Before explaining our local spectral approach for finding overlapping communities, it is necessary to make clear what we mean by *local spectra*.

In traditional spectral clustering methods, one finds the first few singular vectors of the Laplacian matrix[7] of a graph G with n vertices. Suppose the first d singular vectors are obtained, one can form an $n \times d$ matrix $\mathbf{V}$ as a latent space. Then one associates with each vertex a point in this latent space whose coordinates are given by the entries of the corresponding row in the matrix. Vertices are clustered using some method such as k-means clustering algorithm. This method is not likely to work well if the communities are small and heavily overlapping with each other.

We make two fundamental changes to this method. The first modification is to overcome the drawback of computing the singular vectors. Intuitively, the vertices around the seed members are more likely to be in the target community, thus a random walk serves as a natural subroutine to reveal these potential members.

We start a random walk from several known members in the target community and run for a few steps. The number of random walk steps should be long enough to reach out to the vertices in the target community, but not long enough to spread out to the entire graph. Instead of considering a single probability vector, we consider the span of a few dimensions of vectors after the short

[7] In the literature, several different definitions of graph Laplacian exist. Readers can refer to [138] for more details, which serves as a good introductory paper on spectral clustering.

random walks and use it as the approximate invariant subspace (*local spectra*) $\mathbf{V}$. The second is to handle the overlapping situation. Now suppose we wanted to find all the nodes in the same community as node i, it is equivalent to find rows in $\mathbf{V}$ that are nearly identical to that correspond to node i. In other words, we want to find rows in the invariant subspace that point in nearly the same direction as the seed node i. To do so, we look for a sparse vector in the span of $\mathbf{V}$ such that i is in the support. This is equivalent to solve the minimum 0-norm problem

$$\begin{aligned} \min \quad & \|\mathbf{y}\|_0 \\ \text{s.t.} \quad & \mathbf{y} = \mathbf{V}\mathbf{x}, \\ & \mathbf{y} \geq \mathbf{0}, \\ & \mathbf{y}_i \geq 1, \end{aligned}$$

where $\mathbf{y}$ can be viewed as the linear combination of basis vectors in $\mathbf{V}$, weighted by the element in an unknown vector $\mathbf{x}$.

In general seeking sparse vectors in a given subspace is a hard problem. We will use 1-norm vector $\|\mathbf{y}\|_1$ as a proxy for the minimum 0-norm vector, and solve the linear programming problem instead

$$\begin{aligned} \min \quad & \|\mathbf{y}\|_1 \\ \text{s.t.} \quad & \mathbf{y} = \mathbf{V}\mathbf{x}, \\ & \mathbf{y} \geq \mathbf{0}, \\ & \mathbf{y}_i \geq 1. \end{aligned}$$

In the following, we give a formal description of our local spectral approach LEMON for detecting target communities from a small seed set. Given the input

of a set of few vertices $\mathcal{S}$ that are already known to be in the target ground truth community C^*, our algorithm would output the algorithmic community C such that the F1 measure for scoring the similarity between C and C^* is maximized.

Step 0. Subgraph sampling:

In practice, the unknown members in the target community are more likely to be around the seed members, and are usually a few steps away from the seeds. This observation motivates us to reduce the complexity by taking only a portion of the graph into consideration. Ideally, this partial graph should contain as many vertices in the target community as possible, and maintains a small size of the same scale as that of the target community.

To sample the graph, we expand the seed set using random walk. After a few steps of the random walk, vertices with large probability are more likely to be in the target community while vertices with small probability being reached would be treated as redundant ones. If the target community exists for the seed set, then according to [10], this target community would serve as a bottleneck for the probability to be spread out. It is worthwhile noting that other expansion methods such as breadth-first-search (BFS) would entirely ignore the bottleneck defining the community and rapidly mix with the entire graph before a significant fraction of vertices in the community have been reached. In the following, we use $G_{\mathcal{S}} = (\mathcal{V}_{\mathcal{S}}, \mathcal{E}_{\mathcal{S}})$ denote the subgraph extracted from the neighborhood surrounding the seed set $\mathcal{S}$.

Step 1. Generate the local spectra:

Consider the subgraph graph $G_{\mathcal{S}}$ extracted from the neighborhood surrounding the seed set $\mathcal{S}$. Let $\bar{\mathbf{A}}_{\mathcal{S}} = \mathbf{D}_{\mathcal{S}}{}^{-1/2}(\mathbf{A}_{\mathcal{S}} + \mathbf{I})\mathbf{D}_{\mathcal{S}}{}^{-1/2}$ be the normalized adjacency matrix of the graph. We define the normalized adjacency matrix $\bar{\mathbf{A}}_{\mathcal{S}}$ of the graph $G_{\mathcal{S}}$ as

$$\bar{\mathbf{A}}_{\mathcal{S}} \overset{\text{def}}{=} \mathbf{D}_{\mathcal{S}}{}^{-1/2}(\mathbf{A}_{\mathcal{S}} + \mathbf{I})\mathbf{D}_{\mathcal{S}}{}^{-1/2}, \tag{2.4}$$

where $\mathbf{A}_{\mathcal{S}}$ and $\mathbf{D}_{\mathcal{S}}$ denotes the adjacency matrix and the diagonal degree matrix of G, respectively. Consider a random walk starting from exemplary vertices in $\mathcal{S}$. Let $\mathbf{p_0}$ denote the initial probability vector where the total probability is evenly distributed among the seed members. We describe how to efficiently construct the local spectra by iteratively transforming the orthonormal basis starting with a *Krylov subspace* defined below.

Definition 1. *The order-l + 1 Krylov matrix generated by the matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ and vector $\mathbf{p}_0$ is defined by the probability vectors in l successive random walks*

$$K_{l+1}(\mathbf{A}, \mathbf{p}_0) = \Big[\mathbf{p}_0, \mathbf{A}\mathbf{p}_0, ..., \mathbf{A}^l\mathbf{p}_0\Big]. \tag{2.5}$$

The column vectors of the Krylov matrix can be orthogonalized, and form the basis vectors of the *Krylov subspace* $\mathcal{K}_{l+1}$. In other words, the Krylov subspace is defined by

$$\mathcal{K}_{l+1}(\mathbf{A}, \mathbf{p}_0) = \text{span}\left(\mathbf{p}_0, \mathbf{A}\mathbf{p}_0, ..., \mathbf{A}^l\mathbf{p}_0\right). \tag{2.6}$$

The main idea of Krylov subspace is to approximate the original eigenvector problem of size n by one of dimension l + 1, typically much smaller than n. Krylov method finds the largest a few eigenvectors of a large matrix in an iterative way [156], which avoids expensive matrix-matrix operations. We may

expect the basis vectors of $K_{l+1}(\mathbf{A}, \mathbf{p}_0)$ to give good approximations of the eigenvectors corresponding to the $l+1$ largest eigenvalues of $\mathbf{A}$, especially when the dominant eigenvalues decay fast enough.

In Algorithm 1, we briefly summarize the procedure of calculating the local spectral subspace [8] from a specified seed set $\mathcal{S}$. We start by calculating the initial invariant subspace $\mathbf{V}_{0,l}$, which is the orthonormal basis of $\mathcal{K}_{l+1}(\mathbf{A}_{\mathcal{S}}, \mathbf{p}_0)$. And the local spectral subspace can be then obtained by iterating the process specified in LINE 4-6 of Algorithm 1. Figure 2.1 shows an example local spectral subspace $\mathbf{V}_{3,3}$, generated from a synthetic graph with Erdős-Rényi $G(n, p)$ model. In the $G(n, p)$ model, a graph is constructed by connecting nodes randomly. Each edge is included in the graph with probability p independent from every other edge.

Algorithm 1: LOCALSPECTRAL($G_{\mathcal{S}}, \mathcal{S}$)

Input: subgraph $G_{\mathcal{S}}$, subspace dimension l, and random walk step k

Output: local spectra $\mathbf{V}_{k,l}$

1: Compute normalized adjacency matrix $\bar{\mathbf{A}}_{\mathcal{S}}$ using (2.4)

2: Initialize $\mathbf{p}_0$

3: $\mathbf{V}_{0,l} = \texttt{orth}(K_{l+1}(\bar{\mathbf{A}}_{\mathcal{S}}, \mathbf{p}_0))$

4: **for** $i = 1, ..., k$ **do**

5: $\quad \mathbf{V}_{i,l}\mathbf{R}_{i,l} = \bar{\mathbf{A}}_{\mathcal{S}}\mathbf{V}_{i-1,l} \triangleright \mathbf{R}_{i,l} \in \mathbb{R}^{n \times l}$ is obtained by QR factorization so that $\mathbf{V}_{i,l}$ is orthonormal.

6: **end for**

7: **Return** local spectra $\mathbf{V}_{k,l}$

---

[8] In the experiments on real datasets, we fix the walk step k and dimension l to be 3 and 3 respectively. For LFR benchmark datasets, we adopt all together 6 combinations for the (step, dimension) tuple: $(2,3), (2,4), (2,5), (3,3), (3,4), (3,5)$ and the highest F1 score among these combinations will be returned.

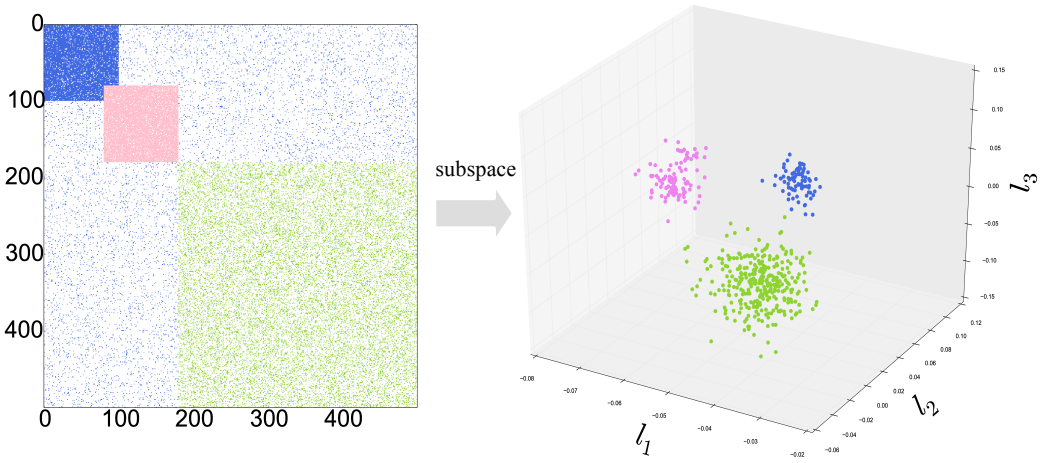


Figure 2.1: An example of local spectral subspace $\mathbf{V}_{3,3}$. The synthetic subgraph G_s is generated with Erdős-Rényi $G(n, p)$ model with background noise $p = 0.05$. The spammer group A and B (denoted by blue and pink respectively) are of size 100 with edge probabolity $p = 0.9$, with partial overlapped 20 nodes. The non-spammer group C (denoted by the green color) has size 320 with $p = 0.2$. The subspace is generated by Algorithm 1 starting from the seed with index 10 in the spammer group A.

Step 2. Seek for a sparse vector

With the local spectra $\mathbf{V}_{k,l}$, we solve the following linear programming problem,

$$\begin{aligned} \min \quad & \|\mathbf{y}\|_1 \\ \text{s.t.} \quad & \mathbf{y} = \mathbf{V}_{k,l}\mathbf{x}, \\ & \mathbf{y} \geq \mathbf{0}, \\ & \mathbf{y}(\mathcal{S}) \geq 1, \end{aligned}$$

where the first constraint indicates that $\mathbf{y} \in \mathbb{R}^n$ is in the span of $\mathbf{V}_{k,l} \in \mathbb{R}^{n \times l}$. The element in $\mathbf{y}$ indicate the likelihood for the corresponding vertex belong to

the target community, which is non-negative. The third constraint enforces that seeds are in the support of sparse vector $\mathbf{y}$, where $\mathcal{S}$ is the set of indices for the seed nodes that we require to be in the community. The entries in the sparse vector $\mathbf{y}$ corresponding to the seeds should be no less than 1.

After sorting the elements in $\mathbf{y}$ in non-ascending order and getting a vector $\hat{\mathbf{y}}$, the vertices corresponding to the top $|C|$ elements in $\hat{\mathbf{y}}$ are returned as the detected community with respect to the seed set $\mathcal{S}$.

Step 3. Reseeding

Augment the initial seed set by adding the vertices corresponding to the top t elements of $\hat{\mathbf{y}}$. Denote the augmented seed set as $\mathcal{S}'$. Then repeat step 1 and step 2 using the augmented seed set $\mathcal{S}'$. The detection accuracy can be improved through iterations via increasing t by a constant number s each time. We define s to be the seed expansion step, which is used as a tunable parameter for adjusting the convergence rate. Usually, the larger expansion step would result in lower performance but a faster running speed with less iterations. In the experiments, we fix the seed expansion step to be 6 for both synthetic and real datasets. The number of iterations for the seed expansion is determined by the stop criteria (Section 2.4.4).

2.4.2 Parameter Sensitivity

The random walk step k, subspace dimension l and seed expansion step s are the key parameters in the local spectral clustering algorithm. We conduct parameter sensitivity study for these three parameter on the four real datasets.

Subspace dimension

To study the parameter of subspace dimension l, we fix the random walk step to be 3, and vary the number of dimension l from 1 to 15. Figure 2.2 (left panel) shows that changing the dimension l does not cause significant fluctuation of the performance on Amazon dataset. On one hand, choosing a large dimension l is undesirable because it would increase the computation cost in the step of generating local spectra. On the other hand, when dimension degrades to $l = 1$, the standard deviation of F1 score becomes significant, making the detection accuracy unstable. Figure 2.3 (left panel) shows that increasing l cause the performance drop on LFR datasets. Therefore, choosing a small value is more desirable considering both the computation cost and performance. In this paper, we fix $l = 3$ because the experiment suggests that setting $l = 3$ can statistically achieve both high and stable performance. Note that such observation holds not only for Amazon network, but for the remaining real datasets as well.

Random walk step

To investigate how the step of random walk affects the algorithm performance, we fix the dimension l to be 3, and vary the random walk step k from 1 to 15. Figure 2.2 (right panel) shows that the average F1 score plateaus as k increases,

and 3-step random walk can yield the algorithm's full potential. The standard deviation, however, significantly increases when k exceeds 10. This indicates that longer random walk is undesirable for stably uncovering the local community structure. Throughout the paper, we fix the random walk step $k = 3$ for the real datasets[9].

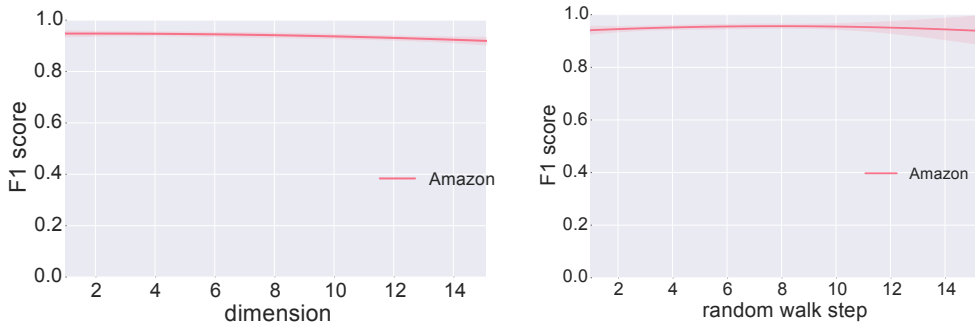


Figure 2.2: The average F1 score on Amazon network with varying dimensions l and random walk step k, respectively. The plots depict the statistical regression line with a 95% confidence interval.

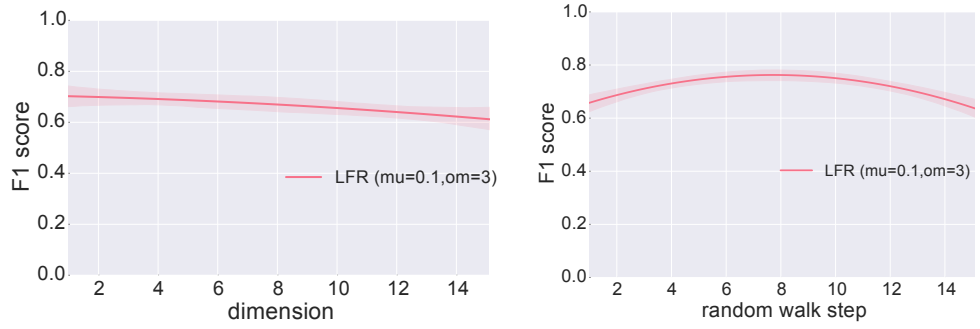


Figure 2.3: The average F1 score on LFR benchmark graph ($\mu = 0.3$) with varying dimensions l and random walk step k, respectively. The plots depict the statistical regression line with a 95% confidence interval.

[9]For LFR benchmark graphs, we adopt all together 6 combinations for the (step, dimension) tuple: $(2, 3), (2, 4), (2, 5), (3, 3), (3, 4), (3, 5)$ and return the highest F1 score among using these combinations.

Seed expansion step

To study the effect of seed expansion step s, we fix both the dimension l and random walk step k to be 3, and vary the expansion step s from 2 to 16. Figure 2.4 shows the results on Amazon (left panel) and LFR (right panel) datasets, respectively. In general, we find our approach is robust to the parameter s when s is kept to a small enough range (e.g., $s \leq 10$). A large incremental step when reseeding is less desirable for stably uncovering the local community structure. Throughout the paper, we fix the expansion step $s = 6$.

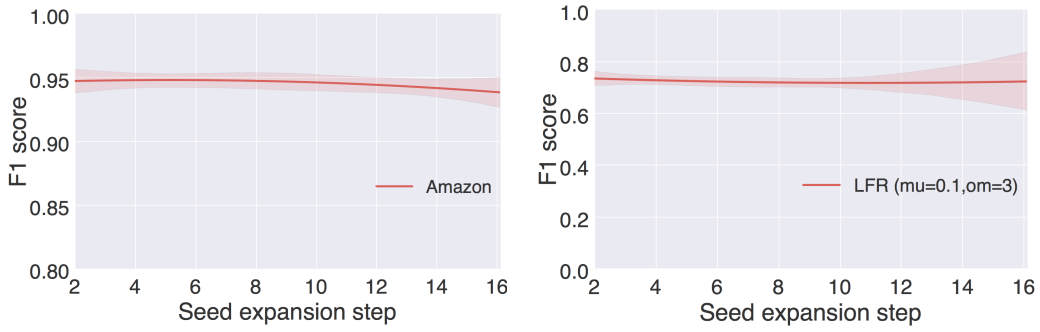


Figure 2.4: The average F1 score with varying seed expansion step s. The plots depict the statistical regression line with a 95% confidence interval.

2.4.3 Complexity Reduction by Sampling Method

If one wants to uncover a small community within a large network consisting of billions of vertices, it would be very costly to take all the vertices into account. We want to discover the target community accurately while keeping the number of vertices examined small. Sampling method can effectively solve the memory consumption issue when one wants to find a local community within a large

graph, since the whole graph does not have to be stored in memory.

In the experiments on real datasets, we conduct a random walk starting from the seed set until the probability has been spread out to $\alpha \cdot |C|_{\text{avg}}$ vertices, where α is some constant and $|C|_{\text{avg}}$ is the average community size in the graph[10]. Note that $\alpha \cdot |C|_{\text{avg}}$ should be large enough to be able to cover as many vertices in the ground truth community as possible. This newly obtained subgraph will be used for the remaining computation. The complexity of our algorithm now depends on the size of the subgraph after sampling, which is $O(|C|_{\text{avg}}{}^{\tau})$ for some small constant τ.

Dataset	**Coverage ratio**	**Sample rate**	$\|C\|_{\text{avg}}$	**Subgraph size**
Amazon	1.00	0.0087	39	2913
DBLP	0.98	0.0076	251	2409
YouTube	0.66	0.0033	79	3745
Orkut	0.64	0.0011	83	3379

Table 2.4: Statistics of the mean values for the sampling method on real datasets.

Table 2.4 gives the statistics after applying sampling method to the real networks. For example, in DBLP network, setting α to be around 10 would yield a subgraph containing on average 98% vertices in the ground truth community. After sampling, we only need to deal with a subgraph of size around 2400 instead of 317,080, bringing a significant reduction of both temporal and spatial complexity.

[10] A fast implementation method for updating the probability vector of the random walks is featured in detail in [10], Section 4.

2.4.4 Round Diffusion Vector via Sweeping Cut

If there are ground truth community sizes available, the above algorithm is guaranteed to stop within few iterations since the seed set will no longer augment once its size exceeds that of the ground truth community. The algorithm would then return the community found with the highest F1 score during the iterations as the result. However, in real case, we don't know the exact size of the communities, causing the ambiguity for most locally based detection algorithm to decide when is the proper time to terminate expanding such that the discovered community is a "good" community. It is thus important to solve the two issues: 1) how to automatically determine the size of the community given a seed set $\mathcal{S}$, and 2) when to stop growing the seed set during the reseeding process.

Determine the size of the community

It has already been shown that random walks produce communities with conductance guarantees and ensure a small boundary defining a natural community in locally based detection algorithms [10]. The intuition is that adding irrelevant vertices to the target community would inevitably cause the conductance to increase, and finding a low-conductance community could ensure the closeness between the detected members and the known seed set. A commonly adopted method of rounding the diffusion values into labels is to perform a sweep-cut procedure on the nodes ranked by the diffusion value, with an objective of minimizing the graph cut metric such as *conductance* [9, 127, 184]. As we will see, the local conductance for a small group of vertices in the graph contains valuable information and enables us to design effective stopping cri-

teria for our algorithm. We define conductance using the generalized Rayleigh quotient specified below:

Definition 2. *Let* $\mathbf{x} \in \{0,1\}^N$ *denote the binary indicator vector for the subset* $\tilde{\mathcal{V}} \subseteq \mathcal{V}_s$ *and* $\mathbf{H} \in \mathbb{R}^{N \times N}$ *is any symmetric matrix. The Rayleigh quotient with respect to* $\mathbf{H}$ *is expressed as the quadratic form of*

$$R_{\mathbf{H}}(\mathbf{x}) = \frac{\mathbf{x}^T \mathbf{H} \mathbf{x}}{\mathbf{x}^T \mathbf{x}} \tag{2.7}$$

In particular, conductance of the set $\tilde{\mathcal{V}}$ measures the fraction of edges leaving $\tilde{\mathcal{V}}$ among all the edges incident on $\tilde{\mathcal{V}}$, and can be expressed using a generalized Rayleigh quotient

$$\Phi(\tilde{\mathcal{V}}) = R_{\mathbf{L}_{\mathcal{S}}, \mathbf{D}_{\mathcal{S}}}(\mathbf{x}) = \frac{\mathbf{x}^T \mathbf{L}_{\mathcal{S}} \mathbf{x}}{\mathbf{x}^T \mathbf{D}_{\mathcal{S}} \mathbf{x}} = \frac{\mathbf{x}^T (\mathbf{D}_{\mathcal{S}} - \mathbf{A}_{\mathcal{S}}) \mathbf{x}}{\mathbf{x}^T \mathbf{D}_{\mathcal{S}} \mathbf{x}}, \tag{2.8}$$

where $\mathbf{L}_{\mathcal{S}} = \mathbf{D}_{\mathcal{S}} - \mathbf{A}_{\mathcal{S}}$ is the Laplacian matrix of graph $G_{\mathcal{S}}$.

Now suppose we have a rough estimation of the lower and upper bound for the size of communities in a graph, which we denote by $|C|_{\min}$ and $|C|_{\max}$ respectively. We could modify the original algorithm in the following way.

At step 2, after obtaining the sorted sparse vector $\hat{\mathbf{y}}$, we are hoping to truncate the sorted vector at some point y_g such that all the vertices corresponding to the elements no less than y_g are included in the algorithmic community. The crux lies in that we do not know which is the best position to truncate the vector $\hat{\mathbf{y}}$. To solve this issue, we denote Λ_i as the set of vertices corresponding to the top i elements in $\hat{\mathbf{y}}$. We then sweep over the sets from $\Lambda_{|C|_{\min}}$ to $\Lambda_{|C|_{\max}}$ and calculate the corresponding conductance for each of the sets. In practice, the value of the conductance with respect to varying size would usually change in a non-monotonic pattern that decreases first and then increases later on. We

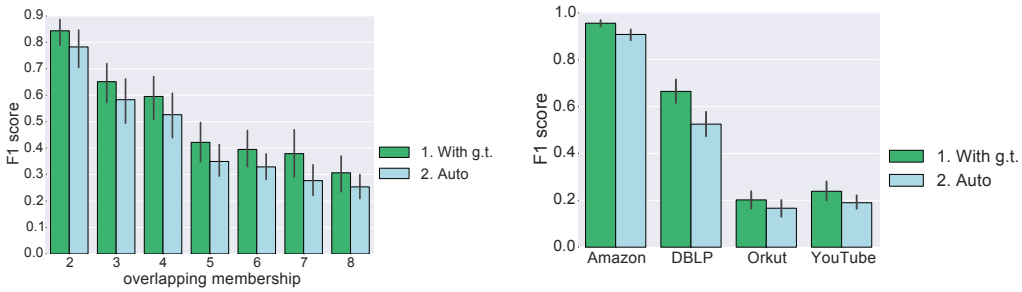


Figure 2.5: Comparison of the average F1 score with ground truth and automatic size determination. The left corresponds to the LFR datasets when $\mu = 0.3$ and the right corresponds to the real datasets.

then adopt the minimum conductance encountered on this curve as the estimated size of the community with respect to the seed set $\mathcal{S}$, which we denote by $\Phi_{\mathcal{S}}^{\min}$.

Stop the reseeding process

As we keep augmenting the seed set through reseeding at step 3, a different seed set would result in a different sparse vector $\hat{\mathbf{y}}$ and thus lead to potentially different algorithmic communities. Practically, one of these seed sets during the augmenting process would achieve the highest F1 score. And it remains to address the issue of when to stop growing the seed set so that it finds the community that resembles most of the ground truth community. This issue can be solved in a similar fashion as that for determining community size. Specifically, we keep track of the value of $\Phi_{\mathcal{S}}^{\min}$ for different seed set during the expansion, and stop to grow the seed set when $\Phi_{\mathcal{S}}^{\min}$ reaches a local minimum and starts to increase for the first time.

Auto detect size vs ground truth size

To verify our method, we compare the performance after applying the stop criteria with that obtained using ground truth communities. Figure 2.5 shows the statistical result of F1 score on both synthetic and real datasets. On both datasets, the F1 score with automatic size determination is only lowered by 10% on average compared with the performance with available ground truth. This implies that our method is applicable for finding communities that mostly resemble the ground truth communities on both synthetic and real datasets in different domains. It also suggests that our method can be applied in practice to uncover natural communities in the situation when no ground truth is available.

2.4.5 Bounding the Performance

In the following discussion, we bound the measure of "tightness" of the extracted community with respect to the subgraph G_S by relating spectral properties to Rayleigh quotients. We start by providing several theorems and lemmas that will be used for deriving the bound of conductance.

Theorem 1. *(Cheeger's Inequality) Let λ_2 be the second smallest eigenvalue of the Laplacian matrix for a graph G_S. Then $\phi(G_S) \geq \frac{\lambda_2}{2}$, where $\phi(G_S) = \min_{\tilde{\mathcal{V}} \subseteq \mathcal{V}_S} \Phi(\tilde{\mathcal{V}})$.*

There are many proofs known for this theorem [39], and we henceforth omit the details here.

Lemma 1. *The generalized Rayleigh quotient $R_{\mathbf{L}_S, \mathbf{D}_S}(\mathbf{x})$ is equivalent to the form of $R_{\bar{\mathbf{L}}_S}(\mathbf{D}_S{}^{1/2}\mathbf{x})$, where $\bar{\mathbf{L}}_S = \mathbf{I} - \mathbf{D}_S{}^{-1/2}\mathbf{A}_S\mathbf{D}_S{}^{-1/2}$ is the normalized Laplacian matrix of graph G_S.*

Proof. By the definition in Equation 2.7 we have

$$\begin{aligned} R_{\bar{\mathbf{L}}_{\mathcal{S}}}(\mathbf{D}_{\mathcal{S}}{}^{1/2}\mathbf{x}) &= \frac{(\mathbf{D}_{\mathcal{S}}{}^{\mathbf{1/2}}\mathbf{x})^T \bar{\mathbf{L}}_{\mathcal{S}} (\mathbf{D}_{\mathcal{S}}{}^{\mathbf{1/2}}\mathbf{x})}{(\mathbf{D}_{\mathcal{S}}{}^{\mathbf{1/2}}\mathbf{x})^T (\mathbf{D}_{\mathcal{S}}{}^{\mathbf{1/2}}\mathbf{x})} \\ &= \frac{\mathbf{x}^T \mathbf{D}_{\mathcal{S}}{}^{1/2} \bar{\mathbf{L}}_{\mathcal{S}} \mathbf{D}_{\mathcal{S}}{}^{1/2}\mathbf{x}}{\mathbf{x}^T \mathbf{D}_{\mathcal{S}} \mathbf{x}} \\ &= \frac{\mathbf{x}^T (\mathbf{D}_{\mathcal{S}} - \mathbf{A}_{\mathcal{S}})\mathbf{x}}{\mathbf{x}^T \mathbf{D}_{\mathcal{S}} \mathbf{x}} \\ &= R_{\mathbf{L}_{\mathcal{S}}, \mathbf{D}_{\mathcal{S}}}(\mathbf{x}). \end{aligned}$$

□

Theorem 2. *(Courant-Fischer Theorem) Let $\mathcal{X}^k$ denote a k dimensional subspace of $\mathbb{R}^N$ and $\mathbf{x} \perp \mathcal{X}^k$ represents that $\mathbf{x} \perp \mathbf{y}$ for all $\mathbf{y} \in \mathcal{X}^k$. For any symmetric matrix $\mathbf{H} \in \mathbb{R}^{N \times N}$ with eigenvalues $\lambda_1^{(\mathbf{H})} \leq \lambda_2^{(\mathbf{H})} \leq ... \leq \lambda_N^{(\mathbf{H})}$,*

$$\lambda_i^{(\mathbf{H})} = \min_{\mathcal{X}^{N-i-1}} \Big(\max_{\mathbf{x} \perp \mathcal{X}^{N-i-1}, \mathbf{x} \neq 0} R_{\mathbf{H}}(\mathbf{x}) \Big) = \max_{\mathcal{X}^i} \Big(\min_{\mathbf{x} \perp \mathcal{X}^i, \mathbf{x} \neq 0} R_{\mathbf{H}}(\mathbf{x}) \Big) \tag{2.9}$$

We will not include the proof of the Courant-Fischer Theorem here. The interested reader can find a proof in any major linear algebra textbook.

We denote by $\mathbf{D}_{\mathcal{S}}{}^{1/2}\mathbf{x} = \mathbf{z}$. With the Courant-Fischer Theorem, we can express the eigenvalues of the normalized Laplacian matrix $\bar{\mathbf{L}}_{\mathcal{S}}$ in the following:

$$\begin{aligned} \lambda_i^{(\bar{\mathbf{L}}_{\mathcal{S}})} &= \min_{\mathcal{Z}^{N-i-1}} \Big(\max_{\mathbf{z} \perp \mathcal{Z}^{N-i-1}, \mathbf{z} \neq 0} R_{\bar{\mathbf{L}}_{\mathcal{S}}}(\mathbf{z}) \Big) \\ &= \min_{\mathcal{Z}^{N-i-1}} \Big(\max_{\mathbf{z} \perp \mathcal{Z}^{N-i-1}, \mathbf{z} \neq 0} \frac{\mathbf{x}^T (\mathbf{D}_{\mathcal{S}} - \mathbf{A}_{\mathcal{S}})\mathbf{x}}{\mathbf{x}^T \mathbf{D}_{\mathcal{S}} \mathbf{x}} \Big) \\ &= \min_{\mathcal{X}^{N-i-1}} \Big(\max_{\mathbf{x} \perp \mathcal{X}^{N-i-1}, \mathbf{x} \neq 0} \frac{\mathbf{x}^T (\mathbf{D}_{\mathcal{S}} - \mathbf{A}_{\mathcal{S}})\mathbf{x}}{\mathbf{x}^T \mathbf{D}_{\mathcal{S}} \mathbf{x}} \Big) \\ &= \min_{\mathcal{X}^{N-i-1}} \Big(\max_{\mathbf{x} \perp \mathcal{X}^{N-i-1}, \mathbf{x} \neq 0} \frac{\sum\limits_{i \sim j} (x_i - x_j)^2}{\sum\limits_i x_i^2 d_i} \Big) \leq 2, \end{aligned}$$

where $i \sim j$ indicates that i is adjacent to j. Similarly,

$$\lambda_i^{(\bar{\mathbf{L}}_{\mathcal{S}})} = \max_{\mathcal{X}^i} \Big(\min_{\mathbf{x} \perp \mathcal{X}^i, \mathbf{x} \neq 0} \frac{\sum\limits_{i \sim j} (x_i - x_j)^2}{\sum\limits_i x_i^2 d_i} \Big) \leq 2. \tag{2.10}$$

Corollary 1. *Given a graph G_S with N nodes, the largest eigenvector of its normalized adjacency matrix is no bigger than 2, i.e., $\lambda_N^{(\bar{\mathbf{L}}_S)} \leq 2$.*

For any symmetric matrix $\mathbf{H} \in \mathbb{R}^{N\times N}$ with orthonormal eigenvectors $\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_N$, and corresponding eigenvalues $\lambda_1^{(\mathbf{H})} \leq \lambda_2^{(\mathbf{H})} \leq ... \leq \lambda_N^{(\mathbf{H})}$, we can always decompose the binary indicator vector $\mathbf{x}$ into a linear combination of the eigenvectors, i.e., $\mathbf{x} = \sum_i a_i \mathbf{q}_i$. This allows us to write

$$R_{\mathbf{H}}(\mathbf{x}) = \frac{\mathbf{x}^T\mathbf{H}\mathbf{x}}{\mathbf{x}^T\mathbf{x}} = \frac{\Big(\sum_i a_i\mathbf{q}_i^T\Big)\Big(\sum_i a_i\lambda_i^{(\mathbf{H})}\mathbf{q}_i\Big)}{\Big(\sum_i a_i\mathbf{q}_i^T\Big)\Big(\sum_i a_i\mathbf{q}_i\Big)} = \frac{\sum_i a_i^2\lambda_i^{(\mathbf{H})}}{\sum_i a_i^2} = \sum_i w_i\lambda_i^{(\mathbf{H})}, \tag{2.11}$$

where $w_i = a_i^2/\|\mathbf{x}\|^2$. Hence, the Rayleign quotient can be viewed as a weighted average of the eigenvalues. If the indicator vector $\mathbf{x}$ forms an acute angle with the invariant subspace associated with the extreme eigenvalues, then most of the weight in the average must be on eigenvalues close to $\lambda_N^{(\mathbf{H})}$. Similarly, $R_{\mathbf{H}}(\mathbf{x})$ can be bounded from below by the smallest eigenvalues of $\mathbf{H}$, in which case the indicator vector $\mathbf{x}$ can be approximated by a linear combination of the eigenvectors associated with the smallest eigenvalues.

Lemma 2. *Let $\mathbf{x} \in \{0,1\}^N$ denote the binary indicator vector for the detected community $C \subseteq \mathcal{V}_s$ corresponding to the seed set $\mathcal{S}$, the conductance of C is bounded by*

$$\lambda_2/2 \leq \Phi(C) \leq \min\{1, 2(1-w_1)\}, \tag{2.12}$$

where λ_2 is the second smallest eigenvalue of Laplacian matrix of G_S, and w_1 is the weight of the smallest eigenvalue of the normalized Laplacian matrix $\bar{\mathbf{L}}_S$, as specified in Equation 2.11.

Proof. The left side inequality holds due to the fact that $\Phi(C) \geq \phi(G_S)$. Following the Cheeger's inequality that $\phi(G_S) \geq \lambda_2/2$ in Theorem 1, we therefore have

$\lambda_2/2 \leq \Phi(C)$. To prove the right side, we first express the conductance $\Phi(C)$ using Rayleigh quotient,

$$\Phi(C) = R_{\mathbf{L}_S, \mathbf{D}_S}(\mathbf{x}), \tag{2.13}$$

which can be further rewritten as $R_{\bar{\mathbf{L}}_S}(\mathbf{D}_S{}^{1/2}\mathbf{x})$, according to Lemma 1. Using similar decomposition as that in Equation 2.11, we can express $R_{\bar{\mathbf{L}}_S}(\mathbf{D}_S{}^{1/2}\mathbf{x})$ as the weighted average of the eigenvalues $\lambda_1^{(\bar{\mathbf{L}}_S)} \leq \lambda_2^{(\bar{\mathbf{L}}_S)} \leq ... \leq \lambda_N^{(\bar{\mathbf{L}}_S)}$, i.e.,

$$\begin{aligned} R_{\bar{\mathbf{L}}_S}(\mathbf{D}_S{}^{1/2}\mathbf{x}) &= \sum_i w_i \lambda_i^{(\bar{\mathbf{L}}_S)} \\ &\leq w_1 \lambda_1^{(\bar{\mathbf{L}}_S)} + (1 - w_1)\lambda_N^{(\bar{\mathbf{L}}_S)} \\ &\leq 2(1 - w_1). \end{aligned}$$

□

2.5 Seeding

Since the initial seed set serves as a key component in our algorithm for uncovering the target community C, it is thus crucial to consider how the quality of seed set affect the performance. In practice, there is not much control over how the seeds are selected. However, the alternative seeding methods can be strategically applied by domain experts in different scenarios based on the availability of candidate seeds. In this section, we will focus on addressing two fundamentally important issues regarding the seed set: 1) What defines "good" seeds? and 2) How many seeds are needed in order to uniquely define a community?

2.5.1 Seeding Method

To give a well-rounded evaluation on this, we encompass in total five different seeding methods here. In this experiment, we adopt $|\mathcal{S}| = 3$ seeds for each of the seeding method listed below.

1. **High degree seeding**: pick $|\mathcal{S}|$ vertices with degree ranked in the top one third among the degree of all vertices in C.
2. **Low degree seeding**: pick $|\mathcal{S}|$ vertices with degree ranked in the bottom one third among the degree of all vertices in C.
3. **Triangle seeding**: pick $|\mathcal{S}|$ vertices in C that form a triangle as the initial seed set.
4. **Random seeding**: pick $|\mathcal{S}|$ vertices in C randomly.
5. **High inward-edge ratio seeding**: the inward-edge ratio for a vertex v is defined by the fraction of links connecting to another vertex inside the target community C among all the links coming out from v. We pick $|\mathcal{S}|$ vertices with inward-edge ratio ranked in the top one third among all vertices in C.

We show the effect of various seeding methods on LFR dataset ($\mu = 0.1$) in Figure 2.6 and real datasets in Figure 2.7, respectively. It is interesting to note that the high-degree seeding method consistently achieves the higher F1 score than low-degree seeding, random seeding and triangle seeding methods. Triangle seeding leads to the worst performance with low F1 score and high standard deviation. This implies that seeding from a compact core structure is less advantageous than seeding sporadically among vertices. The intuitive explana-

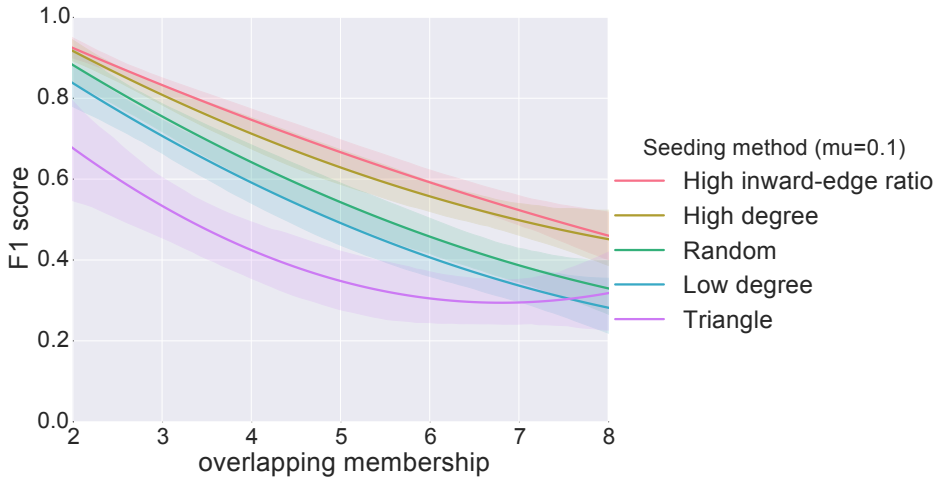


Figure 2.6: The average F1 score on LFR datasets ($\mu = 0.1$) with different seeding methods.

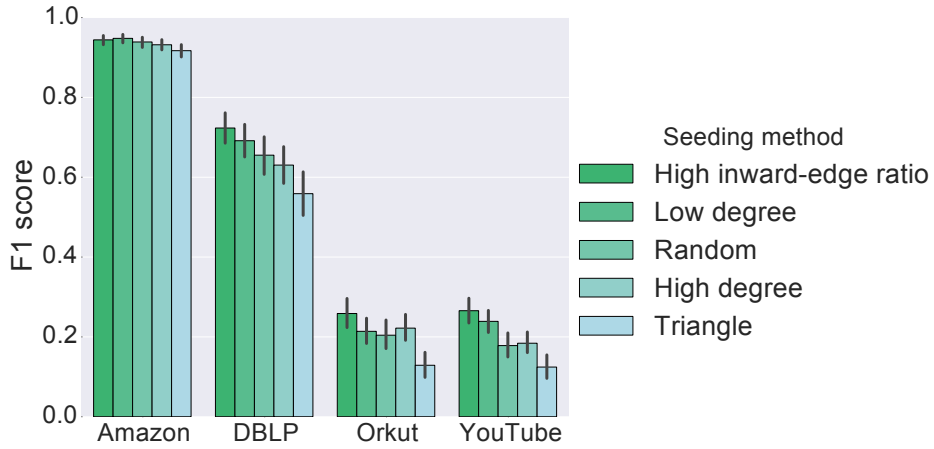


Figure 2.7: The average F1 score on real datasets with different seeding methods.

tion behind this phenomenon is that it is more difficult for the probabilities to spread out when the random walk initiates from a cohesive structure.

Another interesting observation is that high inward-edge ratio seeding method can consistently lead to the best performance among different seed-

ing methods on both synthetic and real datasets. In [95], the authors have the same observation as ours but did not give an explicit explanation on this phenomenon. In fact, when a large fraction of the seeds links connect to vertices within the same community, random walks starting from these seeds would be more likely to transit probabilities into the vertices within the community rather than spreading out to vertices outside the community. A higher detection accuracy can be thus achieved since the target community contains much of the probability after short random walks.

Moreover, it is also striking to note the difference between the test results on synthetic datasets and that on real datasets. Even though the high-degree seeding method can always bring higher performance than that of random seeding on synthetic datasets, the behavior of these seeding methods on real networks is quite different. In Figure 2.7, we see that low-degree seeds lead to better result than that of high-degree seeds on DBLP and YouTube datasets. The degree of seeds does not have a significant impact on the performance in Amazon and Orkut networks since the performance of high-degree seeding and low-degree seeding almost tie with each other on these datasets. In [95], the authors compared the detection accuracy of PageRank based seed set expansion algorithm with high-degree seeding and random seeding on real networks, and concluded that random seeding method always outperforms high-degree seeding in all domains of real networks. However, we remark here that this observation does not apply to our algorithm as we find that high-degree seeding works slightly better than random seeding on Orkut and YouTube datasets.

For all the remaining analysis, we will be using *random seeding* due to its generality and practicality.

2.5.2 Seed Set Size

It is also interesting to investigate how the size of the seed set affects the performance of our algorithm.

We first experiment on the LFR benchmark datasets with varying seed set size. We choose seed set of size proportional to the size of the target community C. Specifically, we test with five different seeding ratios r: 2%, 4%, 6%, 8% and 10% respectively, and round $r \cdot |C|$ to an integer if it is a fraction. Figure 2.8 shows the F1 scores when $\mu = 0.1$. The algorithm's performance can be improved in general as the seed set size increases. In the case when both mixing parameter and overlapping membership are small, e.g., $\mu = 0.1, om = 2$, increasing the seed set size does not seem to affect the performance significantly, and seed set consisting of a small percentage of vertices are sufficient to discover the target community with high accuracy. This implies that when the structure of a small community is well-defined, our algorithm only needs 2 to 3 seeds to reveal the remaining members in a community of size roughly 100. In general, we use an 8% fraction of the vertices in the target community for the whole LFR datasets.

We then carry out the similar experiment on the real datasets. The result on real networks is interesting because increasing the seed set size has little affect on the performance. Especially, using only 3 seeds can yield almost the same performance as using an 8% fraction of the vertices in the target community as seeds on real datasets.

Our algorithm is thus advantageous to many other seed set expansion algorithms that usually require a higher fraction of vertices to be known. For example, in [95], the authors perform a similar experiment on DBLP network.

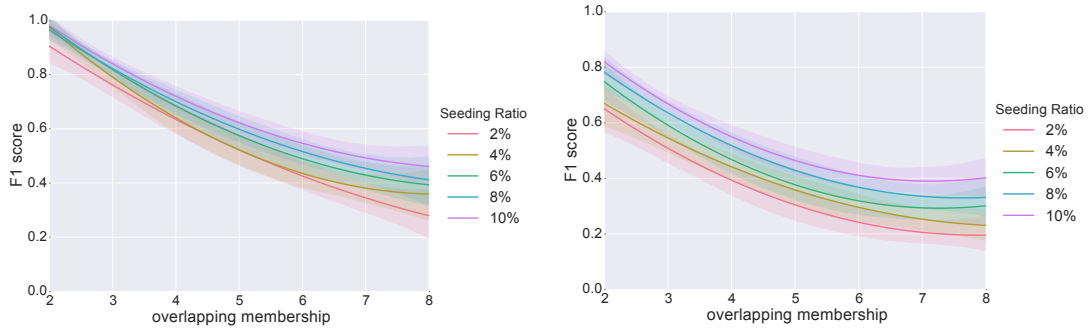


Figure 2.8: The average F1 score on LFR benchmark data with different seeding ratio. The left figure corresponds to the datasets with mixing parameter $\mu = 0.1$ and the right one corresponds to $\mu = 0.3$..

The performance of their algorithm achieves the maximum recall of 0.3 when seeding ratio is 10%, while LEMON can achieve an average F1 score of 0.66 with 3 vertices. This makes our algorithm practical for real networks when it is impossible to collect a large number of seeds.

2.5.3 Further Extension

As the results of using different seeding methods suggests, high-degree seeds can heuristically lead to better result on synthetic data. Such heuristic implies that a vertex with higher degree may exert higher impact on shaping the subspace we are looking for, and thus affect the performance by leading to different sparse vectors where we obtain the "candidates" of the target community from.

In practice, we usually have little control on the seed set. The chance we get a seed set of high-degree members is rare. More often than not, the degree of seeds is randomly distributed. We are therefore inspired to tailor our algorithm

accordingly in order to emphasize the seeds with high degree. The modification is rather straightforward: when calculating the initial probability vector p_0 to start a random walk from, instead of evenly distributing the amount of probability to each seed, we initialize the probability vector according to the degree of each seed. Formally,

$$p_0(v_i) = \begin{cases} d(v_i)/\mathrm{Vol}(\mathcal{S}) & \text{if } v_i \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

where $d(v_i)$ denotes the degree of vertex v_i. In other words, we enforce a bias towards the high-degree vertices at the beginning of the random walk. Note that each time after the reseeding process, the initial probability vector also needs to be recalculated in the same way.

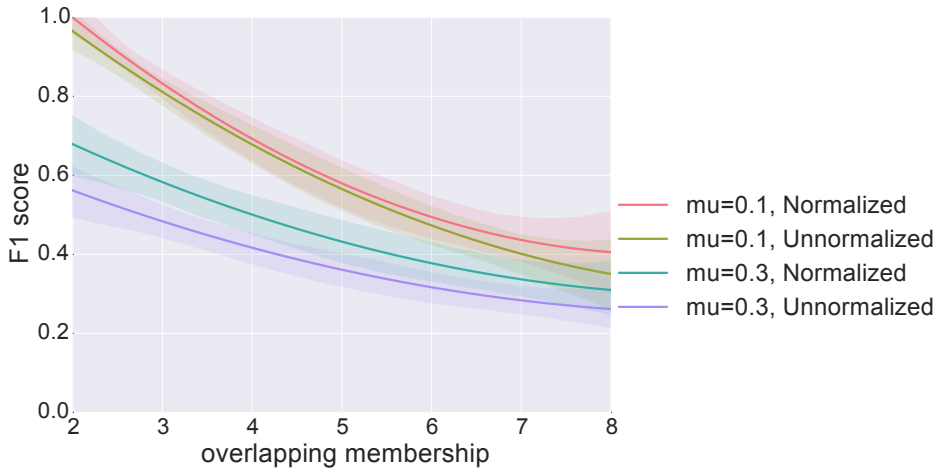


Figure 2.9: Comparison of the average F1 score on LFR datasets with and without normalizing the initial probability vector by each seed's degree.

Figure 2.9 depicts the experimental results on LFR benchmark graphs with and without degree normalized initialization for the random walk respectively. We can find that degree-normalization of the initial probability vector results in

better performance.

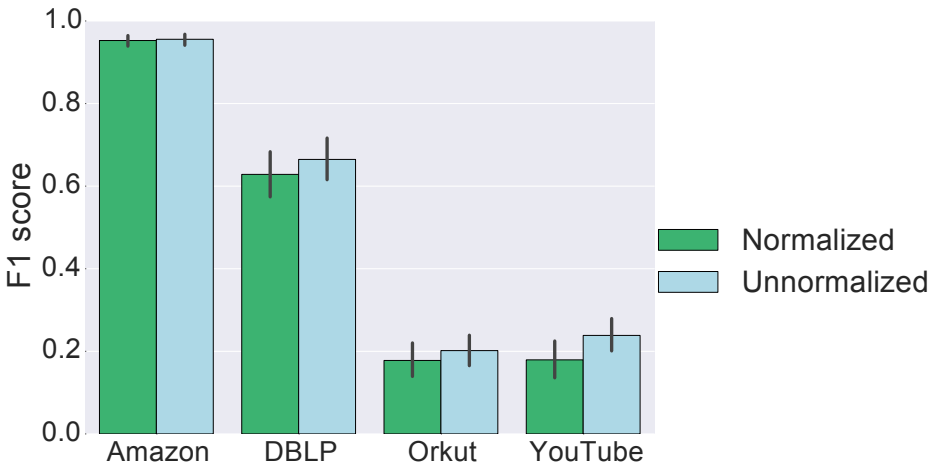


Figure 2.10: Comparison of the average F1 score on real datasets with and without normalizing the initial probability vector by each seed's degree.

We then perform the same experiments on real networks, and find that degree-normalization would on the contrary, lead to slightly worse statistical results (see Figure 2.10). The completely different behavior of using degree normalization on real datasets is rather intriguing. In fact, this phenomenon accords with our previous observation in Section 2.5.1 that a high-degree seed set is less advantageous than random seeds on real datasets. And this explains why emphasizing on the high-degree vertices would worsen the performance on real datasets.

2.5.4 Enlarging the Initial Seed Set

In Section 2.5.2, we see that a larger seed set would lead to better results in general on synthetic datasets. But in the situation when there are not many

seeds available, can we still find a way to improve the performance on synthetic datasets? This can be achieved via preprocessing the seed set before running our algorithm. Specifically, for each pair of vertices (v_i, v_j) in the seed set $\mathcal{S}$, we search for the shortest path $\mathcal{P}$ that connects v_i and v_j, and add the vertices on the path to the original seed set if the length of the shortest path $|\mathcal{P}| \leq 3$. The intuition behind this idea is that any two seeds in the same community must be related for some reason, and they connect with each other either via a direct link or via some other intermediate vertices. In the latter case, those intermediate vertices bridging the seeds are also likely to be in the target community because they serve as the relational "relay" in order for the seeds to be in the same community.

Note that the procedure of enlarging the initial seed set $\mathcal{S}$ differentiates from the reseeding process while running the algorithm. The pre-processing is done before we feed the seed into the algorithm, which is used for the purpose of increasing the size of initial seed set.

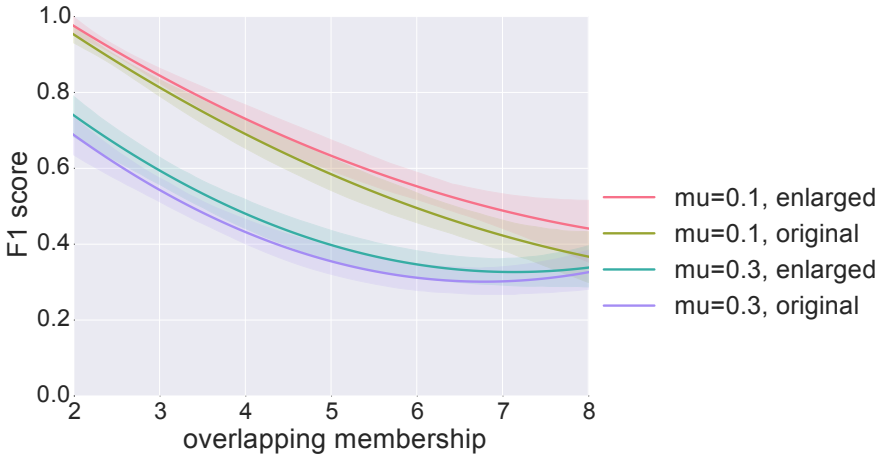


Figure 2.11: Comparison of the average F1 score with and without enlarging the initial seed set on LFR benchmark datasets.

Figure 2.11 presents the experimental results on LFR benchmark data with and without enlarging the initial seed set in advance. We can see that enlarging the seed set can statistically improve the performance. This method can help solve the dilemma of lacking enough available seeds. Especially, this method would help when the seed set consists of 3 or 4 vertices.

2.6 Comparison with State-of-the-art Algorithms

2.6.1 Local Spectra vs. PageRank

The local spectra clustering approach and PageRank algorithm both utilize short random walks to detect the local community structure. PageRank is solely based on the single probability vector, and the latent community members are selected through ranking the probability value among vertices. The local spectral clustering advances PageRank-like algorithms by forming a subspace based on the short random walk, and seeking for a sparse vector such that the seeds are in its support.

	Amazon	**DBLP**	**YouTube**	**Orkut**
LEMON	0.953	0.665	0.240	0.202
PageRank	0.140	0.115	0.136	0.044

Table 2.5: Comparison of the mean F1 score with local spectral clustering and PageRank.

By comparing the performance of these two approaches on the real datasets, we show that seeking for the sparse vector is more effective than directly sorting the probability vector alone. Table 2.5 shows the comparison of average F1 score

obtained by local spectral clustering and PageRank, respectively.[11] From the result, we see that the performance gain brought by the local spectral method is significant, where it achieves more than 5 times higher accuracy on Amazon, DBLP and Orkut networks. We also take into account of a variety of state-of-the-art community detection algorithms for performance comparison in Section 2.6.

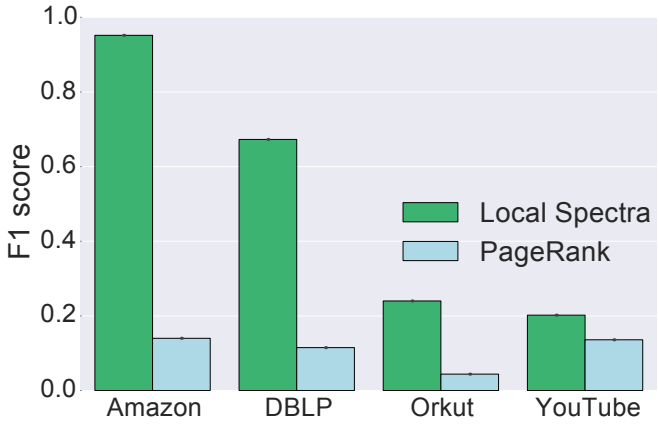


Figure 2.12: Comparison of the average F1 score with local spectral clustering and PageRank algorithms. Height of the bars shows the mean and 95% confidence interval.

To give a well-rounded performance comparison with state-of-the-art algorithms, we further compared our results to three localized community detection algorithms and four global community detection algorithms.

2.6.2 Comparison with Localized Algorithms

We refer to the experimental results reported in some recent publications on localized community detection algorithms [93][95][184]. Figure 2.13 illustrates the

[11] The statistical results of PageRank algorithm is sourced from [93].

comparison of F1 scores on Amazon, DBLP, YouTube and Orkut datasets. We use "LEMON-auto" to denote the results obtained by applying the stop criteria in Section 2.4.4. Since the results on Orkut and YouTube datasets are missing in [184] and [95], we use empty bars to indicate them.

1. **Localized algorithms:** We encompass three locally based methods, Heat Kernel (HK) [93], PageRank (PR) [95] and Seed Set Expansion (SSE) [184].
2. **Heat Kernel** [93]: The heat kernel [12] (HK) is a type of graph diffusion for locally identifying a community nearby a starting seed node. The algorithm can deterministically find the community by computing the diffusion.
3. **PageRank** [95]: The personalized PageRank (PR) scheme is computed using the power method and jumpback probability $\alpha = 0.10$ in [95].
4. **Seed Set Expansion** [184]: The seed set expansion (SSE) approach starts with a phrase of choosing good seed set. The personalized PageRank scheme is then applied to expand the seeds until a community with optimal conductance is found.

Figure 2.13 show that LEMON achieves an F1 score of 0.910 on the Amazon dataset, far outperforming the other algorithms. The average F1 scores increases the performance by 3 times compared with the heat kernel algorithm [93] on Amazon, DBLP and Orkut networks. To compare with [184], we find that the average F1 score of our algorithm doubles their best performance achieved by the "spread hubs" method on Amazon dataset and triples the performance on the DBLP network. The performance comparison of LEMON and PageRank has

[12] `https://www.cs.purdue.edu/homes/dgleich/codes/hkgrow`

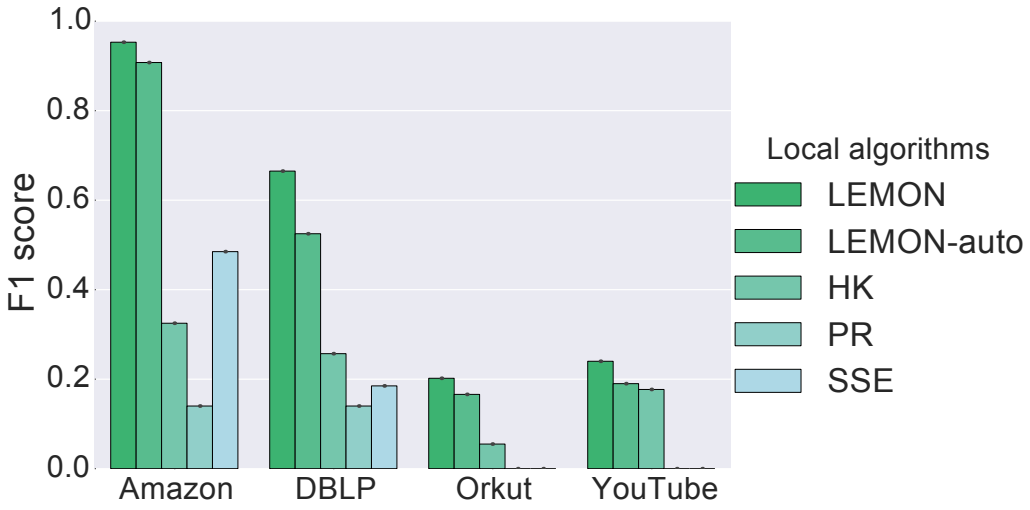


Figure 2.13: Comparison of the average F1 score with state-of-the-art local detection algorithms on real networks.

been elaborated in Section 2.6.1. Also, note that in [95], the authors did not have an explicit stop criterion and instead assumed using a budget for predicting the size of the target community. We compare with the F1 score at a budget of 100 for both Amazon and DBLP datasets. From the results on Amazon networks in [95], we notice that even granted a budget of 400, which is far beyond the average community size of 39 in Amazon network, only a recall of 0.45 can be achieved. And we infer the F1 score would be even lower than this value since the precision is dragged down by the large budget set.

It is also worth noting that we only use 3 randomly picked seeds for all the test cases on each dataset. Our algorithm requires very fewer seeds than other algorithms such as [95].

We further compare the running time with localized random walk based algorithms, namely Heat Kernel [93] and Personalized PageRank [95]. Figure 2.14 shows the running time of different approaches (in log scale) on four

real datasets in consideration. We notice LEMON-auto takes slightly longer time compared to other two walk based approaches, but not substantial. This is due to the component of solving linear programming takes extra computation time.

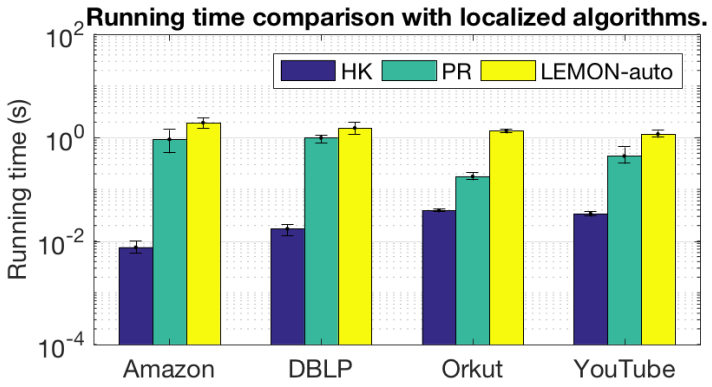


Figure 2.14: Comparison of the running time with local random walk based detection algorithms on real networks.

Algorithm	Implementation	Amazon	DBLP	YouTube	Orkut
LEMON	Python	0.953	0.665	0.240	0.202
LEMON-auto	Python	0.910	0.525	0.190	0.170
DEMON	Python/C++	0.164	0.196	0.031	-
OSLOM	C++	0.766	0.542	-	-
LC	Python/C++	0.815	0.527	-	-

Table 2.6: Comparison of accuracy with global algorithms on real datasets.

Algorithm	Implementation	Amazon	DBLP	YouTube	Orkut
LEMON	Python	<15s	<15s	<15s	<15s
LEMON-auto	Python	<15s	<15s	<15s	<15s
DEMON	Python/C++	4,562s	727,675s	22,395s	-
OSLOM	C++	885,867s	23,262s	>10d	-
LC	Python/C++	4,606s	49,045s	>10d	-

Table 2.7: Comparison of running time with global algorithms.

The experiment has verified that our algorithm is able to achieve high accu-

racy on large networks constituting communities of average size roughly hundred. This implies that our approach is well-suited for the task of detecting small communities in large networks.

2.6.3 Comparison with Global Algorithms

We also compare local spectral clustering with several state-of-the-art global based algorithms.

1. **OSLOM** [109]:

   OSLOM[13] is based on the optimization of a fitness function expressing the statistical significance of clusters with respect to random fluctuations (i.e., the random graph generated by the configuration model [132] during community expansion). The worst case running time of OSLOM is $O(n^2)$.

2. **DEMON** [42]:

   The DEMON[14] algorithm adopts a local-first approach for finding communities. It democratically lets each vertex vote for the communities it sees surrounding it in its limited view of the global system using a label propagation algorithm, and then merges the local communities into a global collection.

3. **LC** [8]:

   Link Community[15] (LC) is a global partitioning algorithm that first builds a hierarchical link dendrogram according to the link similarity and then

[13] `http://www.oslom.org/software.htm`
[14] `http://www.michelecoscia.com/?page_id=42`
[15] `https://github.com/bagrow/linkcomm`

cuts the dendrogram at some threshold to yield link communities. The time complexity is $O(nk_{\max}^2)$ where $k_{\max}$ is the maximum vertex degree in the network.

Table 2.6 and 2.7 summarize the average F1 score as well as the running time of each algorithm on real datasets. Among the baselines, OSLOM and LC fail to terminate within 10 days on the YouTube dataset. The OSLOM algorithm can achieve rather good performance but does not scale well.

In contrast, our algorithm can consistently return the result within few seconds irrespective of how large the entire graph is. Besides, our algorithm has small memory consumption, and a machine with 4GB RAM can afford to process networks as large as Orkut since the algorithm does not have to store the whole graph in memory. Moreover, our locally based algorithm is parallelizable because each seed set expansion can be computed independently. Such property can bring a further performance gain on running time with multi-threaded implementation [184].

Figure 2.15 and 2.16 compares the average F1 score with some state-of-the-art algorithms on LFR benchmark graphs. During the experimentation, we also incorporate the methods that can effectively improve the performance on synthetic datasets that are addressed in Section 2.5.3. We notice that our algorithm outperforms the baseline algorithms even when we use the random seeding strategy. When the mixing parameter $\mu = 0.3$, as is shown in Figure 2.15 and 2.16, LEMON brings about 30% $\sim$ 40% relative improvement compared with the best results among the baselines. And we can expect the performance gain to be even more significant if the seeds possess the qualities discussed in Section 2.5.1.

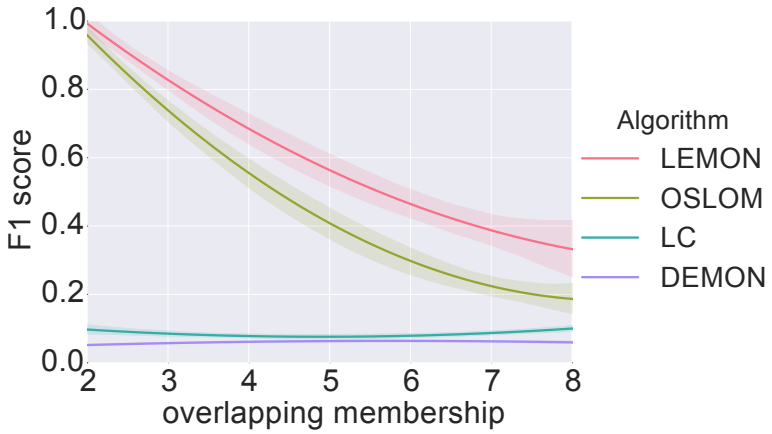


Figure 2.15: Comparison of the average F1 score on LFR datasets ($\mu = 0.1$) with baseline algorithms.

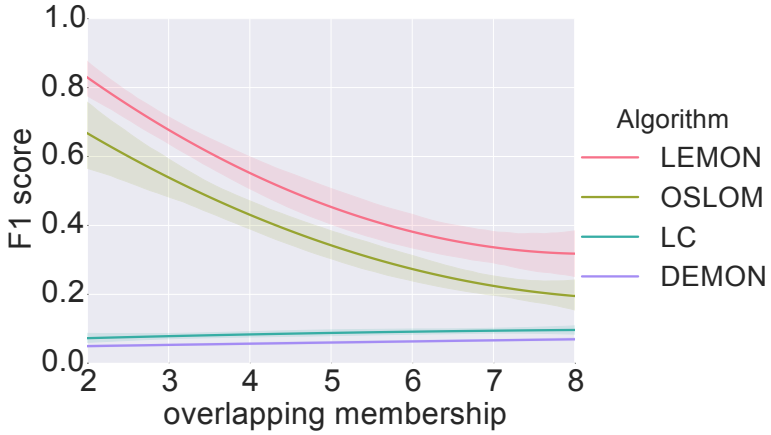


Figure 2.16: Comparison of the average F1 score on LFR datasets ($\mu = 0.3$) with baseline algorithms.

Among the four baselines, we notice that LC and DEMON consistently perform poorly on both groups of the synthetic datasets. We further look into the communities found by LC and DEMON respectively, and find that LC tends to partition the graphs into very small pieces while DEMON, on the contrary, usually finds communities that are much larger than the ground truth communities. This implies that both algorithms extract structures from networks that bear little resemblance to the natural formation of the communities. However, we remark here that even LC fails to recognize the communities well on the synthetic data, it perform better on real datasets as we see in Table 2.7.

2.6.4 Empirical Comparison Between Synthetic and Real Data

Networks are not all similar and we cannot assume one algorithm works for finding communities in a network will behave the same on the other networks. Therefore, it is important to develop the understanding of how different types of networks affect the behavior of algorithms.

Our algorithm sustains a consistent performance on both LFR benchmark graphs and real networks though, we still want to summarize and call the attention to several subtle differences here.

First, LEMON is less sensitive to the parameter of random walk step k and subspace dimension l on real networks than that on LFR benchmark graphs. In practice, fixing (k, l) to be $(3, 3)$ for real networks can ensure a good performance.

Second, LEMON is less sensitive to the seed set size on real networks than that on LFR benchmark. In practice, a seed set size of 3 can guarantee a good

performance on real networks. As for LFR, we adopt the seed set size to be proportional to the community size (8%).

Third, LEMON is more sensitive to the high-degree seeds in graphs where the node distribution is highly skewed (e.g., when the node degree can range from tens to thousands). In LFR graphs, the degree of a vertex is at most 50. Whereas in some large real networks such as YouTube, the degree of some vertices exceeds 1000, making the degree distribution much more skewed than that seen in LFR graphs. And we expect that vertices with unusually high degree in real networks would have a stronger power in controlling the trend for the probabilities to spread out during the random walk, and thus have a higher risk to enter some other neighboring communities. Such an effect can be counterbalanced by putting less initial probabilities on these "super cores".

The above empirical analysis informs us that finding communities in real networks seems to be less parameterized than that on synthetic datasets for our algorithm. This indicates that our algorithm is better suited for uncovering those naturally well-formed communities than the artificially constructed communities in practice.

2.7 Conclusion

The problem of identifying small community structure in large networks has been gaining importance. In this paper, we have presented a method for finding overlapping communities by seeking a sparse vector in the span of local spectra where the seeds are in its support. To overcome the drawbacks of traditional spectral clustering methods, we propose a novel method to construct the local

spectra based on the singular vector approximations drawn from short random walks. Our algorithm enables finding a small community in time functional to the size of the community, and it consistently returns the result within seconds even for a network with billions of vertices. We demonstrate the effectiveness and efficiency of our method for discovering communities on both synthetic and real-world datasets. As the experimental result shows, our algorithm achieves the highest detection accuracy amongst the state-of-the-art proposals.

Many other fundamentally important research questions remain to be addressed. First, the community detection algorithm based on local spectral clustering could be potentially applied to the membership detection problem, i.e., finding all the communities that an arbitrary vertex belongs to. Second, during the process of seed set expansion, we adopt the first low-conductance community as the target community, which usually yields a high resemblance to the ground truth community. It would also be interesting to look further into some larger low-conductance communities and see if a hierarchical structure exists. In this case, some large social group consisting of several small cliques is likely to be discovered.

CHAPTER 3

SCALABLE GRAPH LEARNING: LARGE-SCALE DEPLOYMENT

This section is written in collaboration with Oscar Martinez, Xing Chen, Yi Li and John Hopcroft. The work was published in Proceedings of the 25th International Conference on World Wide Web in 2016.

3.1 Introduction

Every day people generate a large amount of comments on YouTube but not all of those engagement activities are real. Bad actors have been trying to game the system by posting inorganic contents and inflating the count of social engagement metrics.

We consider any practice that attempts to post fake contents, or artificially inflate the number of YouTube engagement metrics through the use of automated means or as a marketplace, as illegitimate activity. Generally speaking, any engagement activity in online social media that does not reflect user's genuine interest can be viewed as *fake social engagement*.

The issue of fake social engagement came into being partly due to that third-party businesses attempt to boost YouTube video engagement metrics in order for promoting contents and increasing popularity. At Google, we have seen attackers attempting to take advantage of the YouTube community by using a variety of deceptive practices [5], including malware, fake accounts, artificial traffic spam and comment spam. Among the various forms of spam activity, fake social engagement has become the most frequently seen yet hardest to de-

tect practice. In particular, we have discovered that abusive YouTube Comments have evolved from traditionally *explicit* spammy-like (e.g. linked with bad URLs or associated with obvious advertisement), to a more *insinuated* outlook that makes them difficult to discern from those organic comments. For instance, one common type of fake YouTube Comments comprises text pieces such as "*cool* ", "*oh*", which have made approaches largely basing on text features and bad URL detection insufficient in such scenario.

We note the importance of keeping the service free of fake engagement activities that may potentially spoil the online social ecosystem. As the YouTube official policy guide on subscription [4] states, for example:

> *Subscribing to a channel creates a relationship between a content creator and a content consumer; the creator keeps making great videos, and the consumer keeps watching, like-ing, and commenting. We take this relationship seriously. A subscription is a user-initiated pledge of support to a YouTube channel; this means that a real human being wants this channel's content in their feed every day. The amount of users subscribed to a YouTube channel should be a metric that reflects genuine interest in that channel, not a gauge of automated or falsified activity.*

And we believe that such policy does not only apply to YouTube Subscribes, but can also be extended to other engagement activities such as Comments as well. As a matter of fact, YouTube is far from the only social media facing the challenge of keeping its service free of deceptive practices. Twitter Followers, Amazon Reviews and Facebook Likes are all buyable by the thousand online [45], for example.

To address these issues, we study the temporal engagement activity patterns on YouTube, making use of anonymized aggregate daily logs of YouTube Comments. We create the engagement relationship graph by taking account the frequency of common engagement activities shared between two individuals within a short period of time. The engagement graph allows us to detect *orchestrated actions* by sets of users which have a very low likelihood of happening spontaneously or organically. Such behavior of groups of users acting together on the same videos or channels at around the same time, is also known as *lockstep behavior* [23].

To detect the lockstep behavior on YouTube, we take a semi-supervised learning approach, making use of existing known abusive accounts as *seeds*. We demonstrate an effective method, LEAS *(Local Expansion at Scale)*[1], in detecting deceitful user engagement based on the local spectral graph diffusion [121]. Local spectral method has substantial advantage over traditional spectral techniques because of its capability in prioritizing and finding clusters *only* near a local region of the engagement graph surrounding the seed. Specifically, LEAS searches for clusters consisting of suspicious nodes with similar pattern of behavior as the given seeds. We show LEAS is scalable to massive datasets, with a straightforward adaption to the MapReduce implementation. Moreover, the MapReduce deployment has the same performance guarantee as the serialization since each diffusion procedure is performed locally. By clustering YouTube users based on their engagement behavior pattern, LEAS can greatly expand the coverage of daily fake engagement take-down volume on YouTube. Our approach can be extended to many other settings including Twitter followers, Amazon product reviews and Facebook Likes etc.

[1]Interestingly, the word "leas" has the meaning of "well-being" in old Irish.

The ultimate goal of our research effort is to help improve social media environment as well as user experience, and to ensure an online world where contents and clicks can be translated into genuine and meaningful interactions. Toward achieving the goal, this paper offers a number of contributions listed in the following:

1. **Problem Formulation:** We provide a novel problem formulation — a semi-supervised learning problem based on the local spectral graph diffusion — to a real-world challenge realized at Google and relevant in many online settings. One advantage of our setting is its full generality. That is, it is applicable for any similarity-based graph without much need for customization.
2. **Algorithm:** We offer a fast, scalable MapReduce implementation adapted from the localized spectral clustering algorithm [121]. This is the first large-scale deployment of local spectral clustering to the best of our knowledge.
3. **Behavioral Analysis:** We show comprehensive performance evaluations on LEAS — focusing on multitudes of different characteristics exhibited by abusive accounts compared to that of general population — using both structural and contextual information.

The remainder of the paper is organized as follows. Section 4.2 describes related work on using graph-based approaches in detecting anomalies. In Section 3.3 and Section 3.4 we mathematically formulate the problem and describe how it can be solved in a semi-supervised learning framework. We introduce the YouTube engagement graph dataset in Section 3.5. A MapReduce implementation is discussed in Section 3.6. Finally in Section 3.7 we offer experimental

analysis, demonstrating the usefulness of our deployment at Google; and conclude our work in Section 4.6.

3.2 Related Work

Online spam activities are evolving as fast as the web services themselves. Abusive actions have been observed in a wide range of domains, including Email [38], web search [19, 35, 141, 181, 185] and blogs [96]. In recent years, spam campaigns have also been prevalently emerging on major social media sites [58, 169], with a diverse set of application targets spanning YouTube [21, 143], Facebook [23, 33, 45], Amazon [123, 134], Twitter [20, 179], eBay [145], and many others.

A number of content-based spam detection strategies have been exploited in the past decade [35, 141, 181, 185]. Most of the proposed methods rely on extracting evidences from textual descriptions of the content, treating the text corpus as a set of objects with associated attributes, and applying classification method such as Support Vector Machine (SVM) [85] to detect spam [71, 141]. A few other more sophisticated methods also take into account the multimedia information such as image features [129, 186].

Content and link based approaches, however, can be infeasible in identifying fake social engagement when contextual information is unavailable, or faking to be organic-like. Many papers tend to devise feature-based classifiers incorporating various account-level as well as social relationship features [21, 169, 193]. While these supervised training models are useful at depicting the spam strategy behind the observed temporal dynamics, it is nonetheless un-

clear how generalizable they are beyond the particular product or signal studied. Furthermore, it is practically difficult to obtain large volumes of training data because manual labeling can be expensive. To this end, recent proposals based on *behavioral clustering* have demonstrated to be effective in spotting groups of users with similar behavior patterns in terms of engagement activities [23, 33, 83, 134, 180, 170]. These methods often start with constructing a bipartite graph representing user-product engagement relationships. Various unsupervised clustering techniques (e.g., co-clustering [23] and community detection [170]) have been applied for detecting groups of actors with similar behavior. Below we highlight a few and illustrate how our work contributes to this line.

More related to our own work, Beutel et al. [23] investigated the problem of fake Page Likes on Facebook and observed a lockstep behavior pattern exhibited by spammers, where groups of users often acting together and Like-ing the same Pages in a loosely synchronized manner. Such collaborative spamming behavior was also observed in Twitter [116] and Amazon product reviews [134], where paid groups of frequent fake review writers have been trying to promote or demote certain products on Amazon. [33] further extended the CopyCatch approach [23] to several other applications such as Facebook app install and Instagram follow. Note that our setting advances [23] by making use of possible domain information in a semi-supervised manner; and our problem formulation is also complementary to [134] which required a large number of both positive and negative examples in a fully supervised manner.

Our work builds on these papers, providing advances in two aspects: algorithmically, our clustering algorithm is operated in a fully localized fashion, which is efficient to compute and easily parallelizable; practically, our frame-

work is fully generalizable, which can be extended to other behavioral clustering problems and applications without much need for customization.

3.3 Problem Formulation

We now describe the mathematical formulation of our problem. We take a semi-supervised approach and define suspicious behavior in terms of graph structure and edge creation times.

To make our problem definition more generalizable, we adopt the notions of *actor* and *target* in representing the entities involved in an engagement activity. For example, in the context of YouTube Comments, a target can be translated into a video.

In the following, we introduce two types of graph that can be created using the engagement activity information. A straightforward way is to build an *engagement bipartite graph* between the set of actors and the set of target, where we use edge to indicate the engagement timestamp. Since we are interested in clustering entities of actors, a more refined way would be to construct an *engagement relationship graph*, in which nodes consist of all the actors and two nodes share an edge if they have acted upon the same target(s).

Mathematically, assuming we are provided with a set of actors, $\mathcal{V} = \{v_i\}_{i=1}^{|\mathcal{V}|}$ and a set of targets $\mathcal{Q} = \{q_j\}_{j=1}^{|\mathcal{K}|}$. We are also given a set of seeds $\mathcal{S} = \{s_r\}_{r=1}^{|\mathcal{S}|}$. Each engagement activity can be described by a tuple of $(v_i, q_j, t_{v_i \to q_j})$, where $t_{v_i \to q_j}$ records the timestamp at which actor v_i acted on target q_j.

- **Engagement Bipartite:** We define $B = (\mathcal{V}, \mathcal{Q}, \mathcal{T})$ as a temporal engagement bipartite graph, where each timestamped edge $(v_i, q_j) \in \mathcal{T}$ records the time at which $v_i \in \mathcal{V}$ acted on $q_j \in \mathcal{Q}$. We further enforce the temporal constraint that all the actors acted on the targets in a $2\Delta t$ time window, i.e.,

$$\exists t_r \in \mathbb{R} \text{ s.t. } |t_r - t_{v_i \to q_j}| \leq \Delta t \; \forall v_i \in \mathcal{V}, q_j \in \mathcal{Q} \tag{3.1}$$

- **Engagement Relationship Graph:** We define $G = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ as a temporal engagement relationship graph. $\mathcal{E}$ is the edge set, where $(v_i, v_j) \in \mathcal{E}$ if actors v_i and v_j have acted upon the same non-empty set of target $\mathcal{Q}_{v_i,v_j} \subseteq \mathcal{Q}$, with weight denoted by $w_{v_i,v_j} \in \mathcal{W}$. Further details regarding the edge weight will be discussed in Section 3.5.

Throughout the paper, our methods and analysis will be focusing on the engagement relationship graph. And we will henceforth use the term *engagement graph* for brevity.

Given: An engagement graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ that models the intensity engagement relationship between nodes; and the seed set $\mathcal{S}$.

Output: Accomplice clusters C_1, C_1,..., $C_{|\mathcal{S}|}$ corresponding to each seed in the set $\mathcal{S}$. Each cluster consists of suspicious nodes with similar pattern of behavior as the given seed, which satisfy the definition of $[n, m, \rho, \Delta t]$-*temporally approximate bipartite core* (T-ABC) given below.

Definition 3. *We define an* $[n, m, \rho, \Delta t]$*-temporally approximate bipartite core (T-ABC) with respect to a given seed* $s \in \mathcal{S}$*, as a set of actors* $C' \subseteq \mathcal{V}$ *associated with a set of*

edges $\mathcal{E}' \subseteq \mathcal{E}$ *such that*

$$s \in C' \tag{3.2}$$

$$|C'| \geq n \tag{3.3}$$

$$|\mathcal{E}'| \geq \rho \cdot \frac{n(n-1)}{2} \tag{3.4}$$

$$w_{v_i,v_j} \geq m \; \forall (v_i, v_j) \in \mathcal{E}' \tag{3.5}$$

Here we introduce the term $\rho \in [0, 1]$ to relax the constraint in the original definition of $[n, m, \Delta t]$-temporally coherent bipartite core (TBC) in [23]. We make such change since we find many loosely connected abusive clusters existing in practice. Relaxing the constraint enables us finding both tightly and loosely connected groups of suspicious actors.

3.4 Semi-supervised learning via local spectral diffusion

We use the semi-supervised learning method to tackle the problem of detecting the suspicious actor groups defined in previous Section. Graph-based learning approach can be viewed as a probability diffusion that propagates large values from a small set of nodes with known labels — which are usually referred to as *seeds* in literature — to the remaining nodes of the graph [60]. This type of approach typically starts with a graph and the labeled sample matrix $\mathbf{S} \in \mathbb{R}^{N \times K}$, where N is the number of nodes in the graph and K is the number of classes. $S_{i,j} = 1$ if node i is labeled with class j, and $S_{i,j} = 0$ otherwise.

A graph-based learning framework usually incorporates two essential parts. The first is to produce an $N \times K$ matrix $\mathbf{Y}$ which *encodes* the probability for each unlabeled node to be in certain classes. Specifically, $Y_{i,j}$ should be large if node i

should be labeled as class j. In our problem setting of binary classification, the diffusion matrix can be reduced to a vector $\mathbf{y} \in \mathbb{R}^N$, where larger value indicates a higher possibility being labeled the same as the seeds. And the second key component is to *decode* the diffusion values in $\mathbf{y}$ into a predicted label based on some graph metric optimization criterion. In the following, we will provide details on both components of our learning algorithm.

Local spectra vs. global spectra

Spectral method is one of the most widely used techniques for exploratory data analysis, with applications ranging from data clustering, image segmentation to community detection etc. Spectral clustering makes use of the first few singular vectors of the Laplacian matrix associated with a graph, which are inherently global quantities and may not be sensitive to very local information [127]. For example, in the case when provided with domain knowledge about a target region in the graph, one might be interested in finding clusters *only* near the specified local region in a semi-supervised manner, which might not be otherwise well captured by a method using global eigenvectors. Therefore, in the semi-supervised setting, our pioneer work on local spectral clustering [121] have substantial advantage over traditional spectral techniques, with the capability of prioritizing and learning more about a local region of the graph surrounding the seeds.

3.4.1 Degree-thresholded Sampling

We apply a degree-thresholded sampling procedure using breadth-first-search (BFS) to get a small subgraph G_s covering a local neighborhood region surrounding the seed. Starting from the given seed s, we take the set of frontier nodes — except for those nodes with degree larger than $d_{\max}$ — into the subgraph node set and repeat the process until the size of the subgraph reaches the specified upper limit N. We enforce the degree thresholding to prevent including extremely high-degree nodes, which are less likely to be spammers[2]. In practice, we choose the parameter of N to be at least several times larger than the maximum size of the cluster of interest $|C_s|$, in order to capture as many nodes in the target group as possible.

3.5 User Engagement Graph

3.5.1 Graph Builder

We create engagement graph by using interactions between users to model the way users interact with a video or a channel. This allows us to detect *orchestrated* actions by sets of users which have a very low likelihood of happening spontaneously or organically.

In practice, the YouTube Comment engagement graph is built with the anonymized aggregate YouTube user activity logs from the past 30 days window, and is updated on a daily basis using a MapReduce implementation. Here

[2]We set $d_{\max}$ to be 500 by default. This is because the degree of most known spammer nodes is smaller than 500, as shown in Figure 3.2.

we take the snapshot of graph created on August 3rd, 2015. The Comment engagement graph consists of hundreds of thousands of nodes and tens of millions of edges. The detailed statistics of the engagement graph in use are not discussed here for privacy reasons. Note that the engagement graph we created here constitutes a subgraph of the entire YouTube engagement graph, where we only captured entities that had activities within the scope of a month.

In the engagement graph, nodes represent users and edges represent common videos or channels in which the users engage. Users that have interacted with a common video will share an edge and are consequently joined in the graph. Edge weights are by default computed based on the number of common engagement activities between two nodes. For example, in the case of users commenting on a YouTube video, this approach translates into users having and edge weight between them equal to the number of common videos they have commented upon.

Adding weight penalty

The way we built the YouTube Comments engagement graph is essentially the same as above except for the subtle difference that node can be two types of entities – a user or a Google+ Page. It is worthwhile noting here that YouTube Comments can be made through the Google+ social platform, without having to log into the YouTube sites. Such feature was powered by YouTube's Google+ comment integration system introduced in November, 2013. Each PlusPage behaves like a unique user ID and can be used to write comments across platforms including YouTube.

In order to detect abuse originating from PlusPages, we add an additional

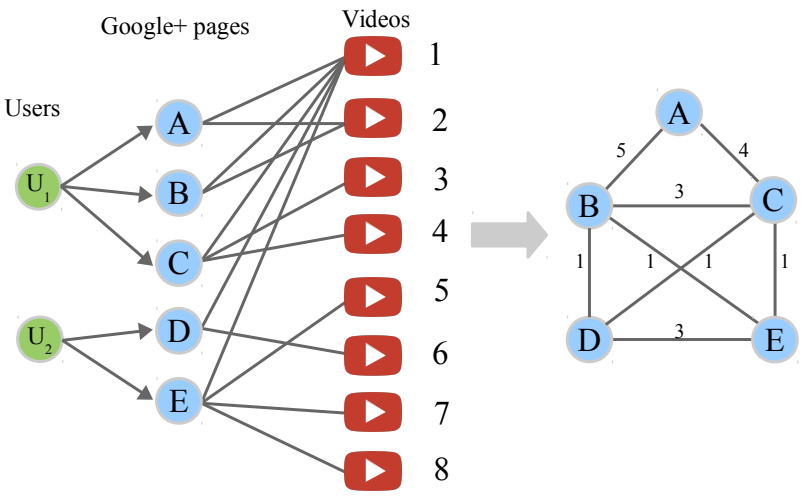


Figure 3.1: Example of constructing Google+ pages engaged graph. It shows a group of two users using their PlusPages to spam video #1.

step when constructing the graph. This modification tends to penalize those PlusPages created by the same user the following way:

$$\tilde{w}_{p_i,p_j} = \mathbb{1}(u(p_i) = u(p_j)) \cdot |\mathcal{P}(u(p_i))| + w_{p_i,p_j}, \tag{3.6}$$

where $\mathbb{1}(\cdot)$ is the indicator function; $u(\cdot)$ defines the owner of a PlusPages and $\mathcal{P}(\cdot)$ gives the set of PlusPages a user has created. We use w_{p_i,p_j} to denote the original edge weight between PlusPages p_i and p_j, and is calculated by the number of common videos both p_i and p_j commented on. $\tilde{w}_{p_i,p_j}$ is the updated edge weight, and is equal to w_{p_i,p_j} when p_i and p_j share different owners. In the case where p_i and p_j are created by the same user, we add extra weight regulated by the total number of PlusPages the user has created. The rationale being that owning a larger number of PlusPages indicates a stronger signal of being potentially abusive.

Figure 3.1 gives an example of constructing Google+ pages engaged graph. It shows that the edge weight between (A, B), (A, C) and (B, C) are all increased by 3, which is the total number of PlusPages the user U_1 has created. The clique structure formed by node A, B and C becomes more noticeable after applying the penalty.

3.5.2 Spammer Seeds

In the context of anomaly detection, when we find suspicious users, we often want to quickly find additional users with similar patterns of behavior that should be disabled as well. LEAS makes use of those users that are identified to be abusive from other YouTube's security mechanisms as seeds.

In practice, spammer seeds are also updated on a daily basis together with the engagement graph. Since the number of available seeds can be limited, LEAS can greatly expand the coverage of daily fake engagement take-down volume.

Degree distribution

We started probing into the behavior pattern between the spammer nodes and the general population by examining the node degree distribution. A salient observation from Figure 3.2 is that the degree distribution of seeds (depicted in magenta) has a dissimilar tail effect compared to that of the general population (depicted in blue). And the difference can be been across all engagement-level activities, and is mostly evident in the Comments graph.

This observation surprisingly corresponds with the fact that spam campaigns and companies involved in selling fake engagements may have efforts

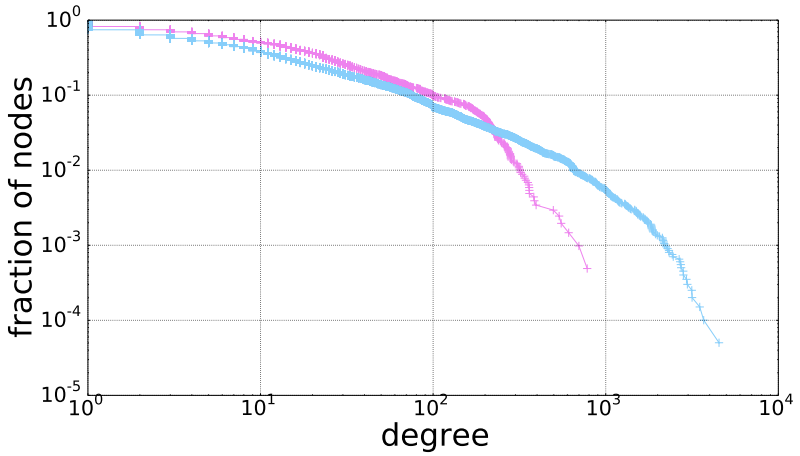


Figure 3.2: Comparison of node degree distribution between spammers and the general population in YouTube Comment engagement graph. The degree distribution of seeds is depicted in magenta, whereas the distribution of general population is depicted in blue. The number of seeds used for plotting is 2k. To plot the general population distribution, we first randomly sampled 10k nodes from the engagement graph. We further excluded those known abusive nodes from the sampled population, which left us with 9,957 nodes. Note that the sampled population may contain unknown malicious nodes.

in relatively modest scope and scale. For example, we looked into several existing online vendor sites that claim to sell YouTube fake engagement. Through investigation we found that YouTube Comments are usually sold with package size ranging from 15 to several hundred, which matches exactly with the seed degree distribution in Figure 3.2. For example, we find spammer nodes rarely have degree greater than 781 in the Comments graph.

3.6 A MapReduce Implementation

Our local spectral diffusion method enables a straightforward adaption to the MapReduce implementation framework. In this Section, we introduce practical details and also potential caveats in applying the method at scale. The implementation is provably scalable to massive datasets and trivially parallelizable, with the capability of searching for many clusters simultaneously. Furthermore, our pipeline has the same performance guarantee as the serialization since each diffusion procedure is performed locally on the graph.

Data Server The engagement graph is served using SSTableService, a distributed in-memory key-value serving system within Google. Each data server holds a partition containing $1/P$ of the total amount of data, where P denotes the number of shards (partitions) of the data. SSTableService allows serving graph queries in a much faster speed compared to on-disk queries. The SSTableService is shared across mappers when running the job.

Data Format We use *Protocol Buffers*[3] for defining the I/O data streams in our implementation. Each protocol buffer message is a small logical record of information, containing a series of name-value pairs. The `graph` protocol namely stores the weighted adjacency list keyed by each node; the `seed` protocol contains the IDs of the spammer seeds; and the `accomplice` protocol defines the output of detected accomplice clusters consisting of suspicious nodes with similar pattern of behavior as the seed. Additionally, we define `config` protocol for conveniently encapsulating and passing configuration parameters to each mapper when initializing the jobs. Some tunable parameters in our pipeline include, for example, the dimensionality of local spectral subspace l, the number

[3] `https://developers.google.com/protocol-buffers/`

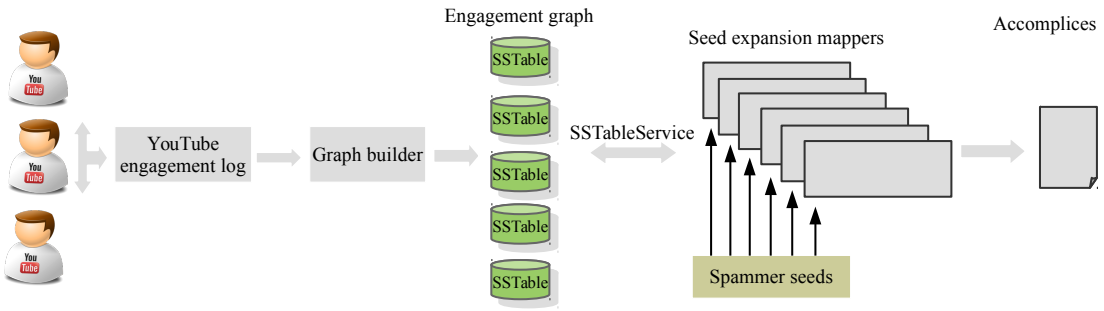


Figure 3.3: MapReduce implementation of YouTube fake engagement detection pipeline.

of short random walk steps k, the minimum cluster size n, the maximum size of the sampled subgraph N, the degree threshold $d_{\max}$ for sampling the subgraph, the edge weight threshold m.

Algorithm 2: MAPREDUCE LEAS

Globals: graph $G = (\mathcal{A}, \mathcal{E}, \mathcal{W})$, configuration parameters

1: INITIALIZEREPLICA()
2: **for** $s \in \mathcal{S}$ **do**
3: **if** deg$(s) \leq d_{\max}$ **then**
4: Sample subgraph G_s
5: $\mathbf{V}_{k,l}$ = LOCALSPECTRAL(G_s, s) ▷ compute local spectral subspace
6: Solve the optimization objective $\mathbf{y}$ in Section 3.3
7: C' = SWEEPCUT$(\mathbf{y})$
8: **emit** $\langle s,$ accomplice $C' \rangle$
9: **end if**
10: **end for**

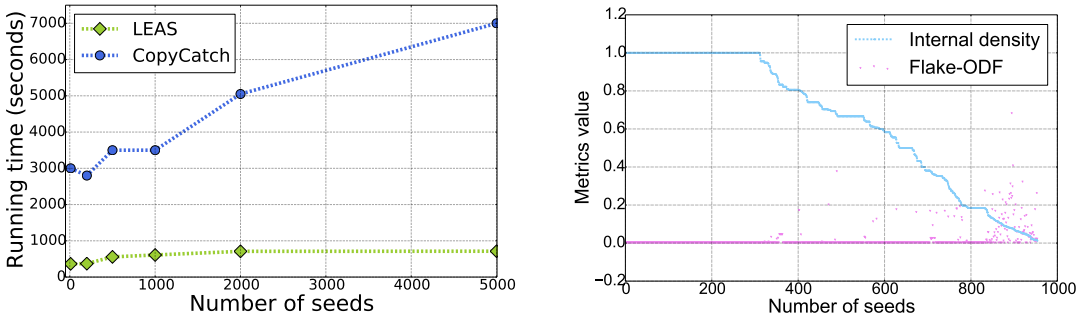
The core of the MapReduce LEAS algorithm can be seen in Algorithm 2. The module of INITIALIZEREPLICA passes the parameters defined by the configuration protocol to all the mappers. And each mapper job processes one seed at a time independently. The entire pipeline of fake engagement detection is illustrated in Figure 3.3, which encompasses the main components of graph builder and seed expander. The graph builder is also implemented using MapReduce framework, where the details are omitted here due to space limit.

3.7 Experimental Analysis

3.7.1 Scalability

YouTube now has over a billion users and is continuing to grow. Therefore, it is important for the algorithm scales well to large datasets in order to efficiently catch the fake engagement activities on a daily basis. We test and compare the performance with COPYCATCH, which is the state-of-the-art algorithm that detects fake Page Likes by analyzing the engagement graph of user-Page interaction.

Firstly, we test the scalability of the algorithm by running our implementation on the YouTube Comments graph over different number of seeds. To make the test results comparable, we choose the same set of seed numbers as that reported in [23]. The number of seeds varies from 100 to 5,000. We additionally run the pipeline with only 10 seeds to test the system starting-up time. Depending on the resources availability, it usually takes about $4 \sim 6$ minutes for



(a) a (b) b

Figure 3.4: (a) Comparison of pipeline running time with state-of-the-art as the number of seeds increases. (b) Internal density and Flake-ODF of detected accomplice clusters in YouTube Comments engagement graph. We filtered those seeds with degree greater than 500, i.e., $d_{\max=500}$ and performed the diffusion algorithm on the rest of the seeds. The number clusters in plot is 955. Cluster indices are sorted by the internal density value.

the system to allocate and set up the data servers and the MapReduce clusters. Figure 3.4(a) shows the comparison of running time between COPYCATCH and LEAS[4]. It is worthwhile noting that LEAS achieves 10 times faster running time with much fewer machines. For example, 3,000 mappers and 500 reducers were used for all the testing data points in [23], whereas at most 1,500 mappers and 2 reducers are required in LEAS test run with 5,000 seeds. Even fewer mappers are required for those tests with smaller number of seeds. For example, running the pipeline with 1,000 seeds uses 295 mappers, 2,000 seeds uses 597 mappers and 10,000 seeds uses 2,999 mappers.

As seen in the results, we find that the running time of LEAS is almost independent of the number of seeds. This is reassuring that our implementation exploits the parallelism of the problem and can continue to scale as the data scales.

[4]We refer to the experimental results originally reported in [23] for evaluation.

3.7.2 Performance Evaluation

Graph Metrics

To evaluate the accomplice clusters found by LEAS, we first measure the structural properties using two commonly adopted metrics [191].

- **Internal density** measures the internal edge density of a node set $\mathcal{V}'$. A larger internal density value indicates a more densely connected community-like structure among nodes.

$$f(\mathcal{V}') = \frac{2|\mathcal{E}'|}{|\mathcal{V}'|(|\mathcal{V}'| - 1)}$$

- **Flake-ODF** is a cluster metric that takes into account both the internal and external connectivity of a set. It measure the fraction of nodes in $\mathcal{V}'$ that have fewer edges pointing inside than to the outside of the set. Ideally, a smaller Flake-ODF value indicates a better cluster quality.

$$f(\mathcal{V}') = \frac{|\{v : v \in \mathcal{V}', |\{(v,u) \in \mathcal{E}' : u \in \mathcal{V}'\}| < \mathsf{deg}(v)/2\}|}{|\mathcal{V}'|}$$

Figure 3.4(b) presents the measurement scores of accomplice clusters detected in three YouTube Comments engagement graph. The most striking observation is the difference concerning the internal density distribution exhibited by the Comments graph. We see that clusters detected from the engagement graph in general are compact with high internal density, which may signify the orchestration strategy when performing fake engagement — that the YouTube fake Comments spammers are exposed to have stronger lockstep behavior pattern, where groups of users acting together, commenting on the same videos at around the same time. The clusters corresponding to the tail part of the curve,

on the other hand, displays a less orchestrated pattern with more likelihood to be incentivized campaigns. Our probe into the structural properties of the detected clusters also suggests that further evaluation is imperative.

YouTube Comment: Manual Review Results

To verify the effectiveness of the algorithm, we ran the pipeline on the engagement graph built on August 3rd, 2015 within 30 days of time window, and performed intensive manual review on the detected accounts. In total, the pipeline detected roughly 24,000 unique accounts with 955 spammer seeds. Among the newly detected accounts, we find that 8,500 of them are found by more than one seed; while the other 15,500 accounts are detected by only one seed. Figure 3.5 depicts the distribution of the frequency for each account being detected by certain seed(s). The fact that an account detected by several seeds is a stronger indication of being potentially abusive. We therefore divide the results into two types and perform analysis accordingly:

- **Tier I:** accounts that are repeatedly detected by more than one seed (35%).
- **Tier II:** accounts that are uniquely detected by only one seed (65%).

To investigate the Tier I accounts, we randomly selected 36 accounts without applying any metric thresholding. We manually examined each account's information and YouTube post history. We also take into consideration the Google internal security measures associated with each account, but will not discuss in detail here for security reasons. The manual review shows that 100% of the Tier I accounts were verified to be fake. Among the Tier I accounts, the most frequently detected account was found by 64 seeds. We find that this particular

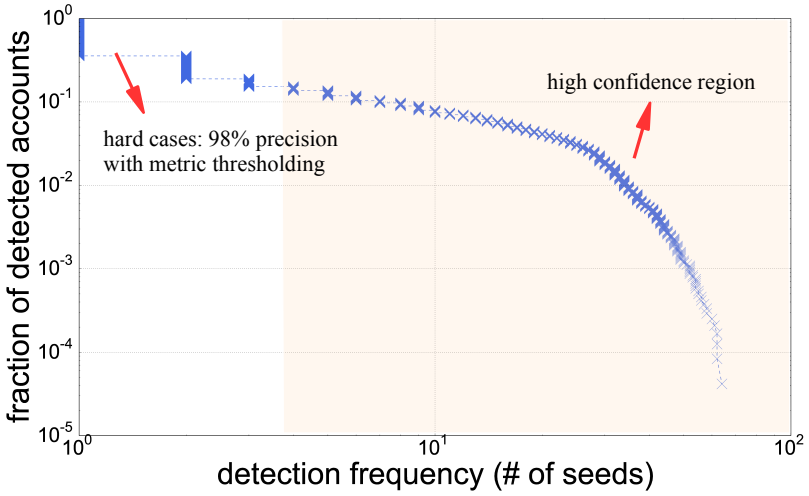
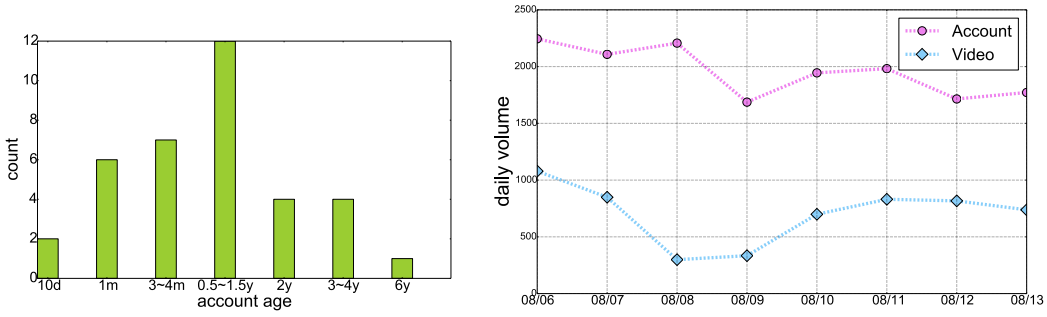


Figure 3.5: Detection frequency distribution of among the accounts detected by LEAS.



(a) (b)

Figure 3.6: (a) Age distribution of 36 manually reviewed Tier I suspicious accounts. (b) Google live runs on YouTube engagement graphs with portion of the seeds, dating from August 6th to August 13th, 2015. The magenta curve depicts the daily volume of unique accounts detected by LEAS pipeline, and the blue curve indicates the daily number of videos these accounts have acted upon.

account was created less than 10 days ago yet had posted more than 253 posts with many quota exceeded. We manually clicked through the comments posted by these accounts, and found that most comments are short text pieces such as "*good videos*", "*very cool* ", "*nice*", "*oh*", "*lol*" or emoji of smile faces. We also find the common pattern for accounts to post exactly the same or similar short, fake comments to different videos. Besides, we also discovered a few accounts posting comments under popular songs, the contents of which are irrelevant to the video content itself but rather asking for view and subscribe (e.g., "*please subscribe*" or "*subscribe now*"). Additionally, several other spammy accounts posting comments including malicious URLs and advertisement were detected.

Besides the contextual information, we also looked into the lifespan of each suspicious account. Although one might expect most spammer accounts to have relatively young age, it was actually quite surprising to see the age heterogeneity of those accounts, as shown in Figure 3.6(a). Among the 36 accounts, the most frequent age falls into the range between 0.5 and 1.5 years; whereas the oldest spammer account have already been existent for more than 6 years.

The Tier II accounts are the harder cases. In order to guarantee the FP guards in production, we randomly selected 100 Tier II accounts that belong to an accomplice cluster with internal density greater than 0.7. The manual investigation shows that 98% detected Tier II accounts to be fake[5]. The comments posted by these accounts share similar pattern as those made by Tier I accounts. Quite interestingly, we indeed found a detected cluster of 15 accounts posting the same comments of either "*i love pets*", "*yeah*" or URLs under certain videos. This further verified that the suspicious groups detected by the algorithm are of high accuracy. As for the other two accounts we are uncertain about, one

[5]In practice, we treat activities made by both Tier I and Tier II accounts as fake engagement.

has huge amount of Google+ shares of good deals although it posted nothing on YouTube comments; another 4-month old account posts a mixture of both organic and fake-like comments, which might be incentivized.

3.7.3 Deployment at Google

LEAS now runs regularly at Google, expanding the coverage of fake engagement activities on YouTube. Parameters have been chosen to significantly distinguish organic user behavior from fake social engagement. There are two levels of take-down actions in practice — engagement level and account level. Engagement level take-down is a soft penalty which removes all the fake engagement activities happened during the day associated with the detected accounts; account level take-down is a more severe outcome, which is applied when we have very high confidence in certain bad actors committing fake engagement from time to time. Figure 3.6(b) shows the daily aggregate volume of detected accounts when running our pipeline on YouTube Comments graphs with *portion* of the spammer seeds, dating from August 6th to August 13th, 2015[6]. We do not display the entire daily take-down volume here for security reasons. Note that engagement level take-down was the main penalty applied during our test runs, henceforth the detected accounts didn't exhibit a fluctuation from day to day — otherwise we would expect to see a decreasing volume of detected accounts when applying the account level take-down policy. Overall, this method, in combination with other existing abuse infrastructure at Google, is effective in decreasing the volume of fake social engagement on YouTube.

[6]The decreased amount of detected account on August 8th and 9th was due to the reduced number of available seeds. In practice, the seeds data is provided by YouTube abuse team and the quantity of which may vary from day to day.

3.8 Conclusion and Extensions

In this paper, we show how fake social engagement activities on YouTube can be tracked over time by analyzing the temporal engagement graph, which models the interactions between users and YouTube video objects. With the domain knowledge of spammer seeds, we formulate and tackle the problem of detecting fake social engagement in a semi-supervised manner — with the objective of searching for individuals that have similar pattern of behavior as the known seeds — based on a graph diffusion process via local spectral subspace. We show our method, LEAS, is scalable to massive datasets, with a straightforward adaption to the MapReduce implementation. We demonstrate the effectiveness of our deployment at Google by achieving a manual review accuracy of 98% on YouTube Comments graph in practice. Our examination on the anonymized YouTube log data also revealed multitudes of different patterns of behavior between abusive accounts and the general population, measured by the average co-engagement intensity, monthly aggregate activity, for instance.

Our approach can be extended to many other settings including Twitter followers, Amazon product reviews and Facebook Likes etc. We envision two future directions towards which our work can evolve. First, while this paper describes the approach in a generic setting, our method can be extended by incorporating other meta signals such as IP address the engagement activities were made from. Second, we believe that better detection model can be derived by taking into account the incentivized engagement behavior, where users are offered incentives (e.g. bonus point rewards) to act on a target such as writing product reviews.

CHAPTER 4

MODELING AND INFERRING ONLINE GROUP DYNAMICS

This section is written in collaboration with Jiezhong Qiu, Jie Tang and John Hopcroft. The work was published in Proceedings of the 25th International Conference on World Wide Web in 2016.

4.1 Introduction

The advent and proliferation of social instant messaging services have been shaping and transforming the way people connect, communicate with individuals or groups of friends, bringing users diverse and ubiquitous social experiences that traditional text-based short message service (SMS) could not. For example, WhatsApp is the most globally popular messaging service with more than 900 million monthly active users (MAUs), WeChat, the largest messaging service in China, has more than 600 million MAUs. These tools have enriched the way people interact by including images, video, location information, audio and text messages. More importantly, they have also catalyzed the formation of social groups, bringing people a stronger sense of community and connection compared with traditional text messaging [40].

While past work has extensively studied the dynamics of group formation and evolution, much of the work is limited to the setting of online communities embedded within the social networking sites — which is inherently different from groups seen in the context of social messaging. Previous study [40] has shown that, for most social messaging tools adopters, the creation and use of

instant group messaging occurs more frequently and habitually than other form of group-level social engagement in their daily life. In terms of lifecycle, social messaging groups have a relatively shorter life span — ranging from several hours to months — as opposed to those online groups seen in social networking sites such as Reddit [31] and Facebook [146] that can sustain up to years. Furthermore, all the chat groups are by default only visible to the group members and grow in an invitation-only fashion, i.e., new members invited to the group are guaranteed to be on the fringe of group networks (one-hop neighbors of current group members) — thus the membership cascade process is more locally dependent, with unidirectional contagion dominated mostly by the existing group members. This is very dissimilar from the diffusion and growing models in previous literature on online communities (e.g., [17, 144]), in which users can make their own decisions to join, even if they are not friends with any of the current group members.

Researchers have recently begun interpreting the group messaging behavior and processes from a social science perspective, yet concrete empirical measurement and statements cannot be drawn from existing literature. Much of the challenge has been the lack of appropriate datasets — one needs a large collection of messaging groups with sufficient time-resolution so that one can keep track of their emergence, growth and demise over time. Another challenge comes from devising an effective model to depict and quantify the diversified, complex processes by which the groups develop over time. As a result, the research community still knows very little about the formation and evolution of chat groups in the context of social messaging — their lifecycles, the change in their underlying structures over time, and the cascade processes by which they develop new members.

To address these issues, in this paper, we analyze the daily usage logs from the WeChat [1] group messaging platform — the largest standalone messaging communication service developed by Tencent in China [3] — with the goal of understanding the processes by which social messaging groups come together, grow new members, and evolve over time. To our knowledge, this is so far the first large-scale analysis on messaging group dynamics. WeChat allows users to send and receive multimedia messages in real-time via Internet. One important feature in WeChat is that any user can create a new group and invite friends to join this group. Please note that such group is invited only, which means that the other users (friends) cannot apply to join if no invitation comes from the group. Groups play a very important role in WeChat. Our statistics show that roughly 25% of the messages in WeChat were generated in group conversations. On the other hand, the groups are very dynamic. Every day, about 2,300,000 new groups were created and about 40% of the newly created groups become silent within only one week. We will describe detailed information about the WeChat dataset along with its mechanics in Section 4.3.

The Present Work: Lifecycle Dichotomy in Social Messaging Groups. In this paper, we contribute to research on group evolution in social messaging platforms by observing and making a conceptual difference between two types of groups in terms of their lifecycle: long-term and short-term groups. Our empirical analysis shows that almost 40% of them stop interaction within one week. On the other hand, we also observe 30% of the groups can survive a much longer period of time ($\geq$ 30 days). The strong lifecycle dichotomy of chat groups leads us to a natural lifecycle modeling and prediction questions — how separable are the long-term and short-term groups by taking into account the

[1] www.wechat.com/en/

structural and social behavioral features? To address this issue, we develop a *separability model* by studying snapshots of millions of groups, and show the strong distinction between long-term and short-term groups — measured with a broad range of features including the underlying group network structure, the membership cascade tree properties (e.g. tree size and depth), and the demographics entropy of group members such as gender, age and region.

We also discuss the phenomena of lifecycle dichotomy from the perspective of the roles and functions that social messaging platforms have in users' daily social experiences. This leads to the question of how does the lifecycle and growth pattern of social messaging groups correlate with the social functions it is serving? It turns out that messaging groups have been commonly adopted as a convenient way of connecting with smaller communities all at once, e.g., a family group, a colleague's group, a classmate's group, as well as groups for social events [40]. And the lifecycle of messaging groups is largely dependent on it social purpose for being setup — for instance, one may expect that event-driven groups will have a higher chance of dying out than friend groups for frequent catching-up.

Furthermore, given the strong separability between the long-term and short-term groups, a fundamental problem concerning the design of successful communities is: Can we predict whether a social group will grow and persist in the long run by analyzing the structural and behavioral patterns exhibited by the group at its early stage? We phrase it as a problem of *early prediction* in group longevity. Through the lens of various features exhibited by a group, we demonstrate that strong prediction results can be obtained even with a group history of one day.

The Present Work: Group Membership Cascade and Prediction. In addition to modeling the growth and evolution from a group-level perspective, we take one step further and investigate the individual-level attributes of group members and study the cascade process by which groups gain new members. Specifically, given the historical behavior of group users as well as the local social structure, can we predict which users in the group are more likely to be active and invite new users to the group chat and to whom will he/she send invitations to? Making sense of such questions requires fine-grained inspection into users' historical engagement behavior as well as the local social network structure that users embedded in. To this end, we develop a membership cascade process model in which we consider features of both *inviter* — a group member who sends invitation to friend(s), and *invitee* — the individual in the inviter's ego networks who gets invited to the group chat. Our inviter prediction model using all features generally achieves AUC as high as 95.31%, and invitee prediction model reaches AUC of 98.66%.

Furthermore, we also attempt to analyze: how does the added new members in return lead to the change of underlying social network structure, as the group evolves over time? To address this issue, we take snapshots and compare the same set of sample group at the timestamp of setup and after a month, respectively. Interestingly, we observe that although both long-term and short-term groups have increment on features such as close triads, long-term groups have shown to increase the close triads more significantly.

Organization. The remainder of this chapter is organized as follows. Section 4.2 describes related work on analyzing group formation and evolution. In Section 4.3, we introduce the WeChat social messaging group dataset. The discus-

sion on group lifecycle dichotomy as well as early prediction model is provided in Section 4.4. The membership cascade process is investigated in Section 4.5. Finally we conclude our work in Section 4.6.

4.2 Related Work

The study of groups and communities is central to many research problems on mining and analytics of sociological data. There have been two major lines of research in this domain: one focuses on the static snapshots of social graphs and seeks to infer and identify tightly-connected group of members — also known as community detection in literature [59, 74, 121, 136, 172]; another line of research focusing on the group dynamics — the growth and evolution of social groups — is more related to our work here. Below we highlight a few and exlpain how ours contribute to the existing research.

Group Dynamics. To understand the process of how social groups form and evolve over time, previous work has extensively investigated the growth and longevity of various forms of online communities such as Facebook apps [94], game communities [49], knowledge sharing communities [192] and social networks communities [17, 32, 86, 144, 172]. A more generalized work of Riberiro [152] investigates the group growth dynamics by encompassing a broad range of 22 membership-based websites.

Our work focuses on studying the group dynamics of a rather understudied realm — the social messaging services. Although researchers have recently begun interpreting the group messaging behavior from a social science per-

spective (e.g., [40]), research community still knows little about the formation and evolution of social messaging groups with concrete and empirical measurement. Focusing on the Yahoo! instant messaging traffic data, Aral et al. [15] considers the individual-level dynamics from the perspective of peer influence and homophily, yet it is unclear how the groups as a whole evolve over time — their lifecycles, their structural dynamics etc. Our work contributes to the current research by discovering a strong dichotomy among social messaging groups in terms of their lifecycle. By taking into account a broad range of group-level structural and behavioral features, we develop a separability model that distinguishes between long-term groups and short-terms groups, as well as an early prediction model that forecasts the longevity of groups.

Cascades. The second half our of work on group membership cascade prediction builds on previous literature that studies diffusion processes [62]. In recent years, scientiests have been able to observe and quantify large-scale diffusions from the richness of online data, including blog space [7, 120], marketing [118], social sites such as LinkedIn [11], Flickr [36], Twitter [61, 106, 154], Facebook [18, 37, 94, 171], LiveJournal [17]. In work that aligns more closely to our focus on social messaging, Aral et al. [15] have looked into the effects of peer-influence and homophily during the diffusion processes. Our work has a more comprehensive scope than [15] by integrating both individual-level and group-level features into our cascade prediction model.

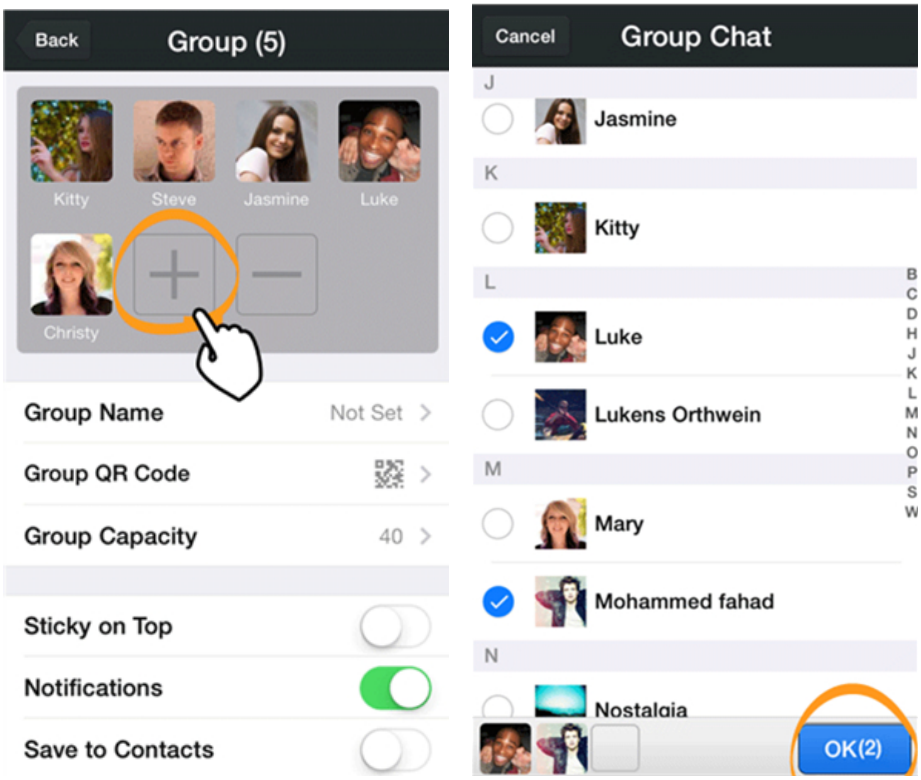
The membership cascade process in the context of social messaging groups differs from previous work in two folds. First, in terms of cascade size, in contrast to previous findings on the structural virality in global-scale cascades [11, 61, 182], we observe a relatively smaller scale of cascade in message groups

and find a siginificant fraction of short-term group memerbship cascades terminate at the depth of 2 or 3. Such difference is caused by the social messaging group nature of maintaining a compact community size than going "viral" [183]. Second, due to mechanics specific to application, the cascade process we consider here is locally dependent, with unidirectional contagion dominated mostly by the existing group members. In WeChat, all the groups are by default only visible to group members and grow in a invitation-only fashion. Only the one-hop neighbors of current group members can be invited to the group chat. And this is very dissimilar to the diffusion models in previous work by Backstrom et al. [17] which assumes that users can make their own decision to join based on the group influence, even if they are not friends with any of the current group members.

4.3 Data

Preliminaries. Before describing the details of the dataset, we first give a brief overview about WeChat's *Group Chat* feature that is central to our study here. While WeChat supports many other important features including *Moments* for photo sharing, *Friend Radar* for searching nearby friends and *Sticker Gallery*, it is important to note that those are beyond the scope of our research focus in this paper.

On WeChat, each user keeps a brief profile, including demographical information (e.g., gender, age and region) and address book which saves the contact list of user's friends. We use the tuple (u, v, T) to denote a friend relationship record if user u becomes friend with user v at timestamp T.



(a) WeChat group membership (b) Membership invitation

Figure 4.1: The WeChat user interaface (UI) of inviting friends to group chat. (a) Group membership UI displays the current users in the group, along with attributes and basic settings for the group such as group name, group capacity etc. Group members can tap the "+" button to invite friends into the group chat. (b) The UI of inviting friends to the group chat. Users can browse and select contacts to add, and click "OK" to send invitations. When group size is under the capacity 40, invited users will be automatically added into the current group chat without requiring further confirmation. However, under the circumstances that group size exceeds the capacity limit, invited users will have to manually click through the invitation message in order to join the group. The largest WeChat group can have as many as 500 members by default. Sourced from WeChat official feature site [2].

A chat group on WeChat can be analogy to a community, where one can chat with several friends all at once. There are two ways in which a user can involve in a chat group: one can either initiate a new chat group, or get invited by an existing member of the group. Figure 4.1 illustrates an example of the WeChat user interface of inviting friends to group chat. We consider (u, v, C, T) as a successful invitation if user v joins group C invited by user u at timestamp T.

After being a member of a chat group, one can send various forms of messages (e.g., text, photo and voice) to the entire group. We use the tuple (u, C, T) to denote the a group chat record if user u send a message to group C at timestamp T.

Data Collection and Cleaning. The data for this study comes from anonymized logs of complete WeChat group messaging activities, collected between July 26th, 2015 to August 28, 2015. We first collect all the 2.3 million groups generated on July 26th, 2015, as our group set of interest. We preprocess the data by ignoring groups with less then 5 chat logs — i.e., we only consider groups that are not born to be dead; and also filtering groups with users that are in list of monthly spam users (MSU) or monthly inactive users (MIU). The list is maintained and updated by WeChat on a monthly basis. All the initial groups in consideration consist of at least three members.

Data Description. After preprocessing the initial group set, we are left with 474,726 groups for further analysis. We then collect four datasets of interest listed below. Tabel 4.1 summarizes statistics of the dataset used for this study.

- **Group Activity Records** $\mathcal{G}$: It consists of all the temporal group activity

records (u, C, T) for each of the sampled group, with T running between July 26th, 2015 to August 28, 2015.

- **User Set $\mathcal{U}$**: It consists of all the members belonging to the sampled groups as well as their one-hop neighbors, as of August 28, 2015. Note that we further remove users in the list of MSU or MIU from the user set.
- **Invitation Records $\mathcal{I}$**: It consists of tuples (u, v, C, T) where user u successfully invites v to join group C at timestamp T during our data collection period.
- **Friendship Records $\mathcal{F}$**: It consists of all the tuples (u, v, T) where u and v $(u, v \in \mathcal{U})$ become friends with each other at time T. The friend relationships in WeChat are undirected, and we have both $(u, v, T) \in \mathcal{F}$ and $(v, u, T) \in \mathcal{F}$.

Table 4.1: Summary of data set.

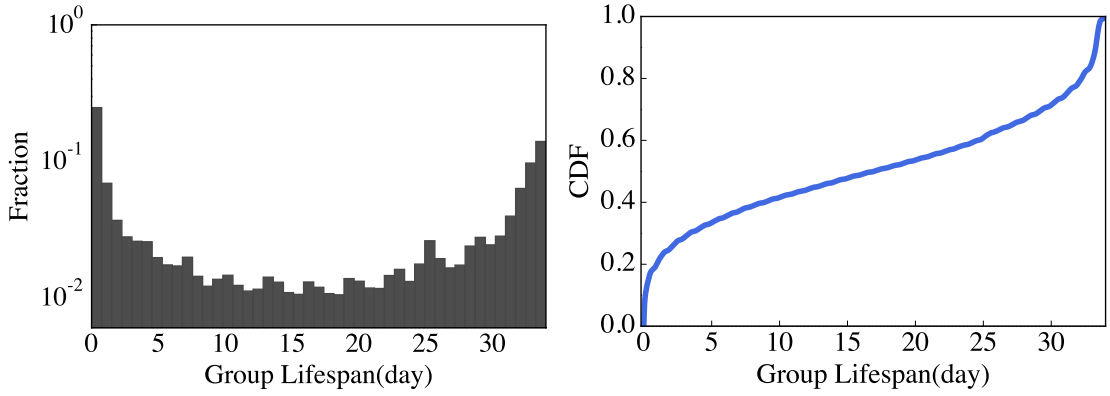
Category	Type	Number
Group	Total	474,726
	Min group size	3
	Max group size	500
User	Total	245,352,140
Invitation	Total	2,013,351
Friendship	Total	624,529,005

4.4 Group Lifecycle Dichotomy

One question that we brought up previously is how social messaging groups grow and evolve over time — their lifecycles and their structural dynamics. As

a high-level characteristic, social messaging groups can have a relatively shorter lifespan — ranging from several hours to months — as opposed to those online groups seen in social networking sites such as Reddit [31] and Facebook [146] that can sustain up to years. In this section, we start with discussing the phenomena of lifecycle dichotomy we observe from the group activity temporal data. To do this, we define the lifespan of a social messaging group below.

Definition 1. Group Lifespan. *We define it by the duration from the timestamp at which a group is initialized, to the timestamp at which no group member sends chat messages anymore.*



(a) Histogram (b) CDF

Figure 4.2: Group Lifecycle Dichotomy. Left: Histogram of ***group lifespan*** (measured by day); Right: Cumulative distribution function (CDF) of ***group lifespan***.

We begin by analyzing the distribution of lifespan among all the 474,726 group samples. Since we stop our data collection on the day of August 28, 2015, the longest lifespan a group can have is 34 days during the period of our observation. Figure 4.2(a) and Figure 4.2(b) display the distribution and Cumulative distribution function (CDF) of group lifespan respectively. A salient observation drawn from the result is that the histogram of group lifespan is dominated by two peaks: one appears on the leftest (near a few hours) and another appears

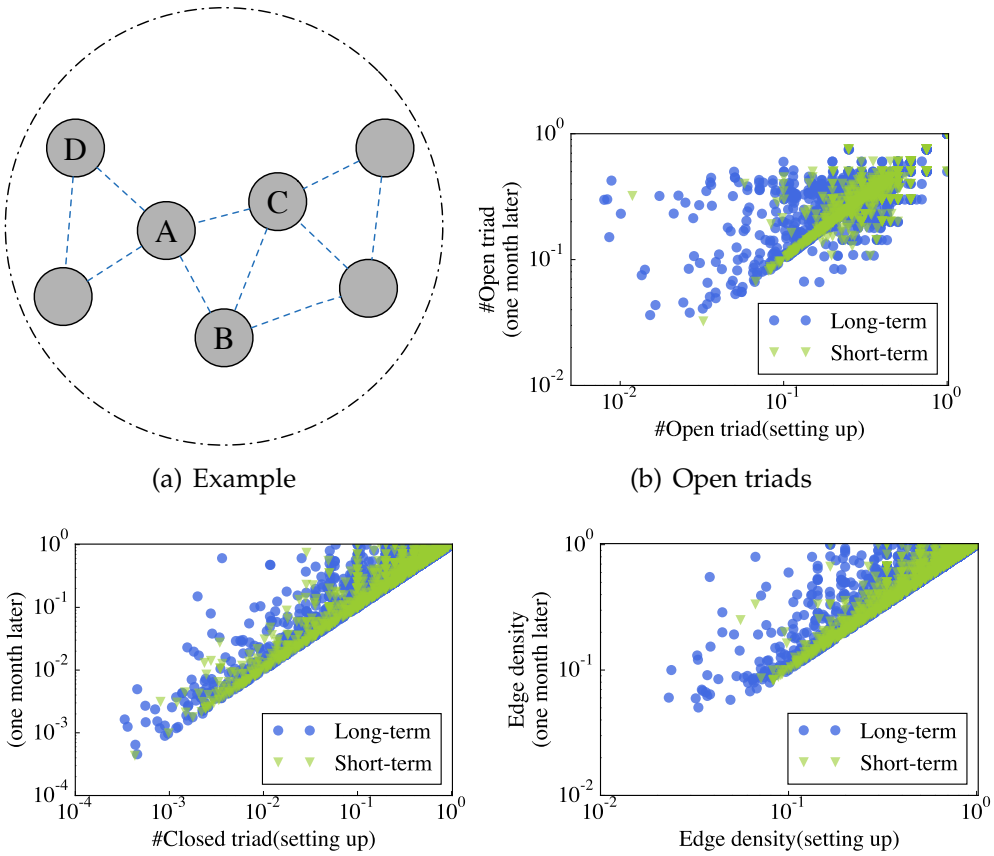
on the rightest side (near one month). This implies a strong dichotomy exists among groups in terms of their lifecycle, and we accordingly make a conceptual difference between two types of groups:

- **Short-term groups**: this type of groups emerge and die very quickly, and usually have lifespan ranging from hours to a few days. For example, Figure 4.2(b) shows that almost 40% of groups stop interaction within only a week.
- **Long-term groups**: this type of groups can survive a much longer period of time than short-term groups. Figure 4.2(b) shows that about 30% groups fall into this category and can sustain longer than 30 days.

The phenomena of lifecycle dichotomy also leads us to the question of how does the lifecycle and growth pattern of social messaging groups correlate with the social functions it is serving? To address this, we manually examine 100 randomly selected groups, among which 60 are long-term groups and 40 are short-term groups, respectively. We categorize these groups according to their social functions (the title of groups) by hand, and list the details in Table 4.2. Quite interestingly, we find that most short-term groups are event-driven (e.g., travel groups, meeting groups and dining groups), while long-term groups are more relationship-driven (e.g., family groups, colleague groups and friend groups).

4.4.1 Group Structure Dynamics

In this subsection, we move on to study the underlying structural change of messaging groups over time. We investigate several representative structural



(a) Example (b) Open triads

(c) Closed triads (d) Edge density

Figure 4.3: Group Structure Pattern. (a): An example of group friendship networks. Group members and friend relationships are represented as nodes and dot lines, respectively. For this example, we can see *A*, *B* and *C* are connected as a closed triad; *A*, *C* and *D* are connected as an open triad. The edge density of this group is 0.476. (b) (c): Horizontal axis is the normalized number of open/closed triads at the setting up of a WeChat group, and vertical axis is the normalized number of open/closed one month later. (c): Horizontal axis is the edge density at the setting up of a WeChat group, and veritcal axis is the edge density one month later.

Table 4.2: Case study by group displayed name.

Category	Long	Short	Example
Travel	0	8	Discuss on a short trip
Meeting	1	2	Schedule an official meeting
Event	4	13	Plan a wedding
Entertain	5	13	Dine together
Organization	9	0	Departments of company
Class	12	4	Course for GRE test
Friend	13	0	Childhood friend
Family	16	0	A family of three

features (e.g., open triad count, closed triad count and edge density), and quantitatively analyze the how these features evolve in a different pattern with respect to the long-term and short-term groups, respectively.

Triad Count. The studies about transitivity in social networks [73] suggest that the local structure in social networks can be expressed by the *triad count*. In WeChat groups, we try to examine whether long-term and short-term groups show different transitivity patterns. We take into account both the open triad count and close triad count, based on the friendship networks structure of sampled WeChat groups. To illustrate this, Figure 4.3(a) shows an example of a small WeChat group friendship networks, in which nodes A, B and C form a closed triad; nodes A, C and D is considered an open triad.

Edge Density. We also consider the feature of internal *edge density* of a group, which is defined by the fraction of edges (friendships) within the group among all the possible edges when the group is fully connected.

To see how these structural features change over time, we take two snapshots for the groups: one at the time when the groups are initialized (we choose ~10 minutes in this study), another after one month being setting up. We consider long-term and short-term groups separately in order to see the different patterns of structural patterns between these two. We also remark here that although short-term groups may stop messaging interactions at some point, members within the groups are still likely to build friendship as long as they maintain the group membership, and thus affect the underlying friendship network structure for potentially longer period of time.

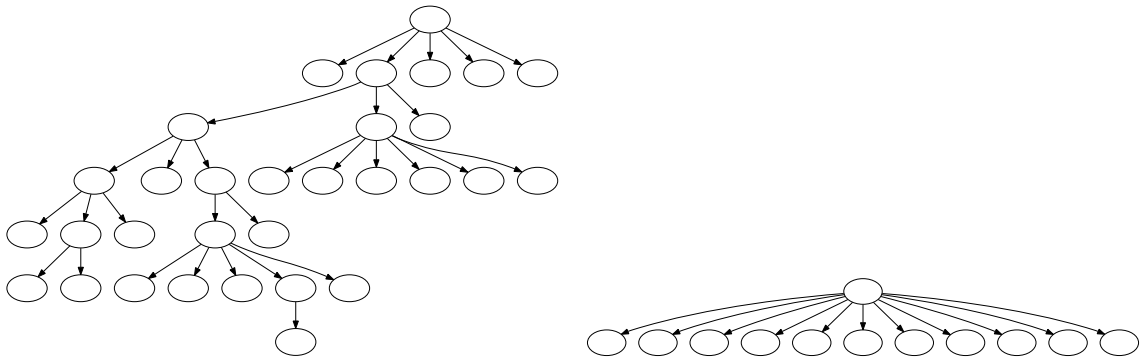
Figure 4.3(b), Figure 4.3(c) and Figure 4.3(d) show the results for feature dynamics of open triad count, close triad count and edge density, respectively. Note that if the structure of groups are not changing at all, we would expect to see a scatter plot centering around the diagonal line of $y = x$ (with normalization). From the visualization results, it is interesting to first observe the different evolution patterns exhibited between short-term groups and long-term groups — the long-term ones show stronger dynamics in terms of the underlying friendship structure features while most short-term groups are less likely to develop friendship over time.

We infer such dichotomy in structure dynamics is related to the social roles and functions for the social groups to be setup. For example, a colleague's group served for long-term communications is more likely to develop social connections between members, as opposed to a group setup for some specific social event.

4.4.2 Cascade Tree Pattern

Beside studying the friendship structure, we also discuss the group formation processes, namely by investigating the group membership invitation cascading tree structure. We start with defining the *group cascade tree* below.

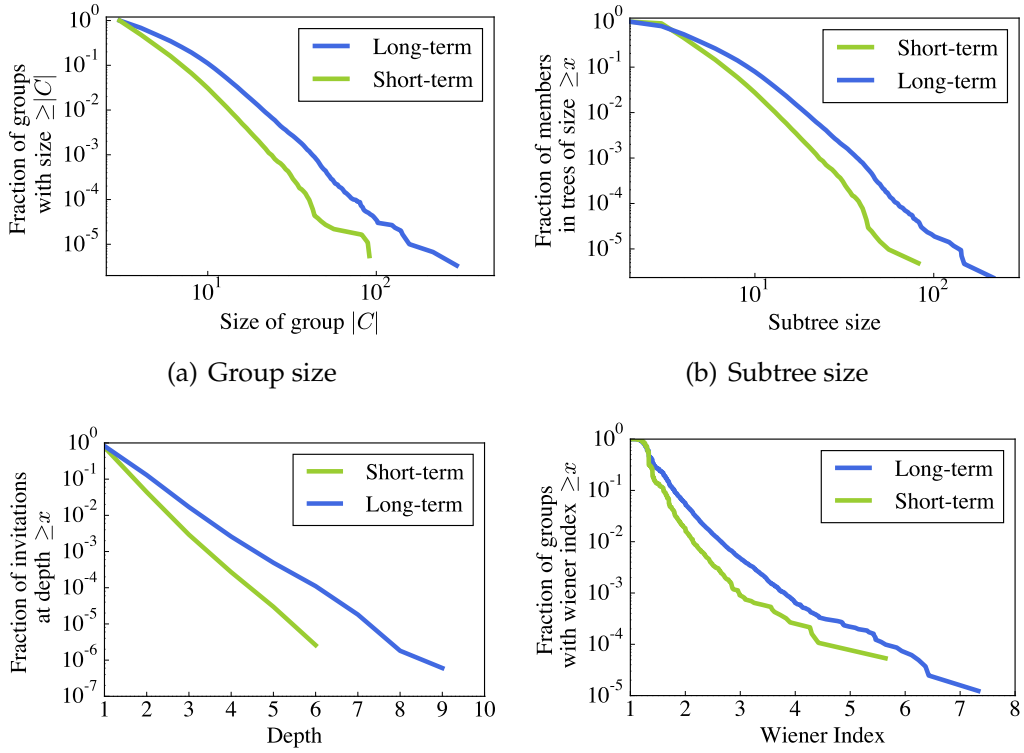
Definition 2. Group Cascade Tree. *A directed graph where each group member is a node, and a directed edge from u to v is constructed if u (inviter) successfully invites v (invitee) to the group. The tree is rooted at the user who initiated the group. Cycles are impossible since inviters always join the group earlier than invitees.*



(a) Example of long-term group (b) Example of short-term group

Figure 4.4: Example of WeChat group cascade tree for long-term group and short-term group, respectively.

To show how long-term and short-term groups differ in terms of cascade tree structure, Figure 4.4(a) and Figure 4.4(b) show the examples for two types of WeChat group cascade tree. We find that long-term groups tend to exhibit a deeper tree structre with more branchings; whereas many short-term group cascade trees display an approximate star graph structure with most members being the leaves of the root node. In order to quantify such difference, we consider here four representative features concerning the structure of cascade trees.



(a) Group size (b) Subtree size

(c) Depth (d) Wiener index

Figure 4.5: Cascade tree related feature distributions. (a): Distribution over group size. Vertical axis is the fraction of groups with size larger than $|C|$. (b): Distribution over subtree size. Vertical axis is the fraction of non-singleton members in trees of specific size. (c): Distribution over cascade depth. (d): Distribution over Wiener index.

Cascade Size. We start our analysis on cascade tree by examining the total number of nodes in the cascade tree, i.e., group size. Figure 4.5(a) shows the normalized distribution of cascade tree size for both types of groups. We find that long-term groups tend to have larger size (up to 500 by default) while the size of short-term groups diminishes around 100. This is not very surprising since long-term groups can be advantageous in gaining more members give a longer timespan for growth.

Invitation as a Function of Cascade Depth. A natural way to measure the dif-

ference in cascade tree between long-term groups and short-groups is to examine the distribution of cascade depth at which the invitation occur. We measure the cascade depth for each of the invitation happens during our observation period, which is defined by the number of steps from the root to the group member in the cascade tree. Figure 4.5(c) shows the normalized distribution of cascade depth among all the invitations in our dataset. We observe that more invitations occurs far from the root in long-term groups than that in short-term groups. For example, 10% of invitations in long-term groups occur at depth 3 or greater; whereas for short-term groups, only less than 1% of invitations occur at depth 3 or greater.

Invitation as a Function of Subtree Size. Finally, we measure the difference of cascade tree structure between long-term groups and short-term groups by measuring the size of subtree for each node in the cascade tree. In Figure 4.5(b), we show the distribution of the cascade subtree size for each node resides in the cascade tree, aggregated among all the sampled groups. Again, we observe substantial difference between long-term groups and short-term groups. For example, 30% of nodes in the cascade tree of long-term groups have subtree size greater than 10; whereas only 10% of nodes have subtree size greater than 10 in short-term groups.

Structural Virality. We also quantify cascade trees by measuring their *structural virality* as that used in [61]. Structural virality, also know as Wiener index, is useful for disambiguating between shallow, broadcast-like diffusion and the deep branching structures. Wiener index is defined by the average distance between any two nodes in the cascade tree. For example, the cascade trees in Figure 4.4(a) and Figure 4.4(b) have Wiener indexes of 3.99 and 1.83, respec-

tively. In Figure 4.5(d), we show the distribution of Wiener index of cascade trees for both long-term and short-term groups. We observe that more than 99% of short-term groups have Wiener index smaller than 2, which implies that most membership cascades happen in a broadcast fashion, settled mostly by the root node.

4.4.3 Group Lifecycle Prediction

The strong dichotomy of group lifecycle and structure dynamics leads us to a natural modeling and prediction questions – how separable are the long-term and short-term groups by taking into account the structural, behavioral as well as demographical features? Can we predict whether a social group will grow and persist in the long run by analyzing the structural and behavioral patterns exhibited by the group at its early stage? In this section, we address both issues through analyzing the snapshots of millions of groups, combining a broad range of features.

Separability Model

In this model, we consider the task of predicting whether a group is long-term or short-term, from features including the underlying group network structure, the membership cascade tree properties, and the demographics entropy of group members. The full list of features can be found in Table 4.5, where we are only using group-level ones for this task.

To train the separability model, we construct the training dataset by labeling

groups with less than 5 days of lifespan as negative examples, and groups with longer than 25 days of lifespan as positive examples. We represent each group as a feature vector extracted one month after groups are built, and further train the dataset using support vector machine (SVM) [54] with 10-fold cross validation.

Table 4.3: Feature contribution analysis on the separability of long-term and short-term groups.(%)

Features used	**AUC**	**Prec.**	**Rec.**	**F1**
All Features	**66.62**	63.23	57.66	60.32
-Structure	64.75	59.36	62.83	61.04
-Cascade	65.36	**64.49**	47.67	54.82
-Demographics	65.24	57.35	**65.71**	**61.25**
Random Guess	50.00	50.00	50.00	50.00
+Structure	**64.21**	61.98	42.51	50.43
+Cascade	61.23	57.35	**65.71**	**61.25**
+Demographics	62.77	**63.18**	41.41	50.03

The prediction results are shown in Table 4.3. We find that highest classification accuracy (66.62% AUC) can be obtained with full set of features. We further investigated how each set of features (i.e., structure, cascade and demographics) affects the training performance by considering only one at a time. And we find that the set of structural features by itself can yield high accuracy, which again confirms that strong distinctions exist between short-term and long-term group structures.

Early Prediction of Group Lifecycle

Given the strong separability between the long-term and short-term groups, we pose a fundamental question of how well can we predict if a group can grow and persist in the long run, from the features exhibited in its early age?

Table 4.4: Group lifecycle early prediction performance results(%). We train the classifier using all the group-level features.

Features used	AUC	Prec.	Rec.	F1
1 hour	57.95	54.16	56.80	55.45
1 day	**65.08**	**61.92**	**53.38**	**57.34**
5 days	65.46	62.52	54.11	58.01
10 days	65.57	62.48	56.81	59.51
20 days	65.76	62.78	56.56	59.51
1 month	66.62	63.23	57.66	60.32

The way we implement the early prediction model is largely similar to the separability model previously except for the subtle difference that group features (see group-level features in Table 4.5) are extracted at earlier timestamps. Specifically, for each group in our training set, we take multiple snapshots at the age of 1 hour, 1 day, 5 days, 10 days, 20 days and 1 month, and calculate the feature vector accordingly. We repeat similar procedure to train the dataset with respect to features extracted at various timestamp, and compare the training performance. Table 4.4 shows the prediction performance results at different stages. We find that features extracted one day after the groups being set up can yield AUC accuracy as high as 65.08%, which is almost as good as the prediction accuracy of 66.62% when adopting features at timestamp of 1 month.

The results of early prediction model reassure that the likelihood for social

messaging group to grow in the future can be well inferred from its very early age (e.g., 1 day). Such predictability is in contrast to previous study on predicting the longevity of online social communities [86] which requires features at the age of *months* for making short-term prediction and *years* for making long-term prediction. And again this is partly due to the different nature of social messaging groups and online communities in terms of the lifecycle.

4.5 Membership Cascade Process

Now we have modeled the growth and evolution of social messaging groups from a group-level perspective. In this section, we approach the problem with a focus on the individual-level and study the membership cascade process by which group gain new members.

To start with, we introduce a group membership cascade model, as illustrated in Figure 4.6. The model captures two important roles: *inviter* — a group member who sends invitation to friend(s), and *invitee* — the individual in the inviters ego networks who gets invited to the group chat[2]. For instance, the big dotted circle in Figure 4.6 encompasses all the current members within a group. There are two essential steps behind each invitation: 1) a member in a group become active (denoted by blue in Figure 4.6), and 2) the active member selects his/her friends (denoted by red in Figure 4.6) into the group chat.

[2]On WeChat, instead of sending group invitation to any registered user, one can only invite his/her current friends into the group chat.

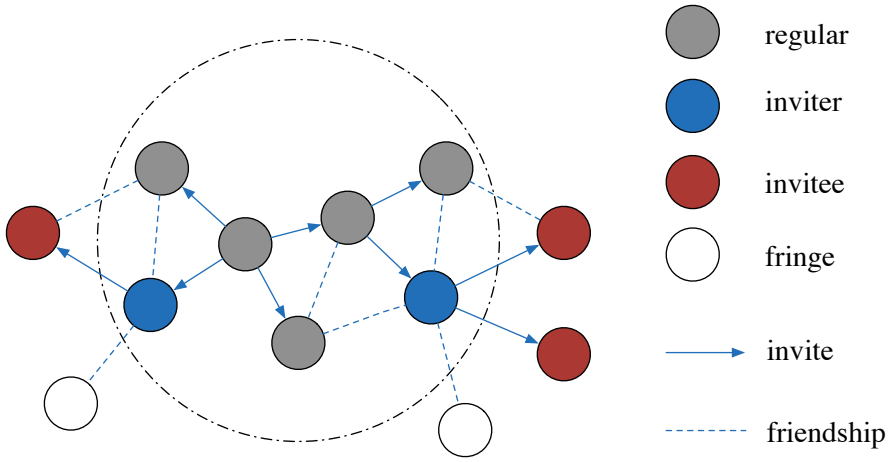


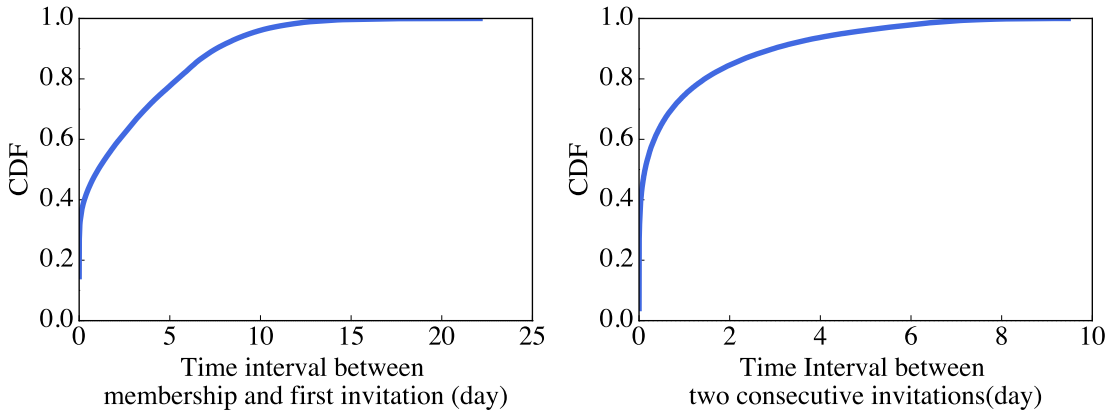
Figure 4.6: Graphical example of WeChat groups' cascade process model. At some timestamp T, some members in a group become active (denoted by blue) and select their friends (denoted by red) to the group chat.

4.5.1 Membership Cascade Pattern

Behavioral Pattern

To have a better understanding of membership cascade pattern, it is important to first study group members' behavioral pattern. For example, an interesting question would be how often do people invite their friends into the group chat once they become a group member? This can also be phrased as how often do membership cascade happen in social messaging groups? In this subsection, we provide some empirical findings concerning members' invitation behavior pattern measured by the concepts of *invitation interval* and *first invitation latency* defined below.

Definition 3. Invitation Interval *is defined as the time interval between any two consecutive invitations from a group member. Additionally,* **First Invitation Latency** *is defined as the interval between the timestamp at which a user joins a group (invited by some existing member) and the timestamp when he/she, for the first time, invites*



(a) First invitation latency

(b) Invitation interval

Figure 4.7: Dynamic pattern of invitations. Left: Cumulative distribution function (CDF) of ***first invitation latency*** (measured by day); Right: Cumulative distribution function (CDF) of ***invitation interval*** (measured by day). The invitations in WeChat group display a highly time-sensitive pattern.

another friend to the same group.

Intuitively, investigating the action of a group member's first invitation is useful since it signifies how well has he/she been adapting to the current group, and how strong the sense of relevance he/she has with respect to the current group.

To address above questions, we obtain the distribution of invitation interval and first invitation latency, aggregating over each member in each group. Figure 4.7(a) and Figure 4.7(b) show the CDF curve of first invitation latency and invitation interval, respectively. We observe the invitations in WeChat groups are highly time-sensitive. On the one hand, when one is invited to a group, he/she tends to start invite other people soon. For example, about 80% of the first invitations happen within 5 days after the inviter joining the group. On the other hand, we find that members suffer from a longer latency in sending their

first invitations than the invitation interval in general. For example, more than 80% of consecutive invitations happen within 2 days of interval.

The Influence of Local Structure

In this subsection, we probe into the local structure of group and investigate how the membership cascade process is influenced by the structural features. Specifically, we study how the probability for a user u being invited by his/her friend into a group is affected by the structure of ego networks of u.

Backstrom et al. Revisit. In the context of online social networks, Backstrom et al. [17] introduced the probability that an individual adopts and joins a community with the number of friends already in the group.

As noted earlier, a significant difference in our setting of social messaging group is that new group members are "passively invited" to the group rather than "actively adopt" the group. To this end, we calculate $P(k)$ — the fraction of user who is invited to a community as a function of the number of k of their friends who are already members, with slightly tweaking the definition in [17]. Specifically, we first take two snapshots of group membership, with 10 days apart. We find tuples (u, C, k) where $u \notin C$ at the time of the first snapshot and u has k friends in C at that time. We then compute $P(k)$ by looking at the fraction of tuples (u, C, k) that $u \in C$ at the time of the second snapshot.

The results for WeChat group (see Figure 4.8(b)) exhibit an interesting contrast to the curves of LiveJournal and DBLP groups shown in Backstrom, Huttenlocher, Kleinberg and Lan [17]. Instead of observing an increasing trend of the curve with respect to larger values of k, we find a qualitatively different

shape where the adoption probability suffers from a slight decrease at moderate values of k, and drastic fluctuations when k exceeds 40. We infer such difference is caused by the mechanism inherent to WeChat, that when a group has more than 40 users, inviting friends to join the group requires their confirmation. We leave the detailed inspection on this for future investigation.

Furthermore, we also observe a strong evidence for the influence of structural locality. In particular when k is small, the fraction of invitee (invited to the group) with 10 friends already in the group ($k = 10$) is twice as much as the fraction of invitee with 5 friends in the group ($k = 5$).

Structural Diversity. The above analysis informs us that the number of friends in the group can affect whether a user gets invited. Yet it is unclear how it is affected by the local network structure. In this subsection, we study how structural diversity of user u's ego networks [176] affects the probability for u to be invited by friends into a group.

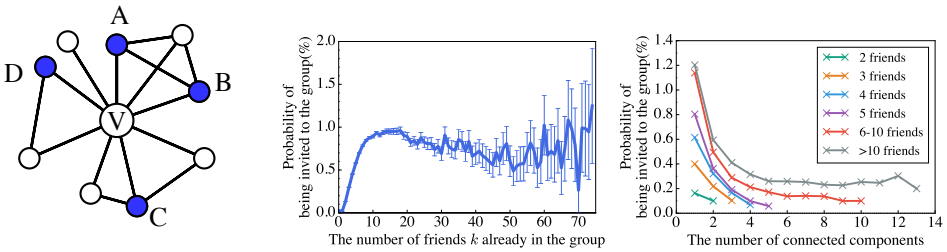
For a given user u with k friends already in a group, we measure structural diversity by counting the connected components in the networks formed by these k friends. For example, as illustrated in Figure 4.8(a), user v has four friends (A, B, C and D) already in a group. A, B, C and D form 3 connected components, i.e., $\{A, B\}, \{C\}$ and $\{D\}$. Figure 4.8(c) plots the curves of the probability of being invited to a group with respect to the number of connected components formed by friends already in group. We choose the parameter k (i.e., the number of friends already in group) to be 2, 3, 4, 5, 6-10 and > 10, respectively. Interestingly, we find that given a fixed k, the more closely these k friends are connected, the more likely u will be invited to the group.

Table 4.5: List of group-level features.

Group Level (for group C at time T)	
Group Structre	Number of individuals with a friend in C (the *fringe* of C)
	Number of edges with one end in the group and the other in the fringe.
	Number of edges with both ends in the group.
	The number of open triads at time T and at the setting up of group.
	The number of closed triads at time T and at the setting up of group.
	Clustering coefficient.
Cascade Tree	Number of Members ($\|C\|$).
	Tenth quantile of members' depth in the cascade tree.
	Tenth quantile of members' subtree size.
	Number of members whose depth equal to k, $k = 1, 2, \ldots, 9$.
	The sum of the lengths of the shortest paths between all pairs of vertices, i.e., Wiener index.
Demographics	Number, fraction of members from country X.
	Number, fraction of members who stated their gender to be X.
	Number, fraction of membes who stated their age to be X.
	Entropy of member's region (country, province, city) distribution, e.g., $-\sum_{x \in \text{Countries}} P(x) \log_2 P(x)$.
	Entropy of member's age: $-\sum_{x \in \text{Ages}} P(x) \log_2 P(x)$.
	Entropy of member's gender: $-\sum_{x \in \text{Genders}} P(x) \log_2 P(x)$.

Table 4.6: List of features individual-level features.

Inviter Level (for member u in group C at time T)	
Historical Behavior	How long has it been since u joined C.
	How long has it been since u invited others to C.
	The number of users that u invited to C before time T.
	The number of chat made by the individual.
Local Structure	The number of u's friends in the fringe with $\geq k$ friends in the group, $1 \leq k \leq 20, k = 30, 40, 50$
	The number and fraction of u's friends in the group: $\|\{v \| v \in ego(u) \ \wedge v \in C\}\|$.
	The number and fraction of u's friends not in the group: $\|\{v \| v \in ego(u) \wedge v \notin C\}\|$.
	The number of cross group edges in u's ego networks: $\|\{(v, w) \| v \in C \wedge w \notin C \wedge u, w \in ego(u)\}\|$.
	The ratio of cross group edges to cross group edges that could possible exist in u's ego networks.
	The depth of u in cascade tree.
Invitee Level (for user u in the fringe of group C at time T)	
Demographics	User u's stated gender.
	User u's stated age.
	User u's stated region (country, province and city).
	The fration of group member who has the same gender as u.
	The fration of group member who has the same age as u.
	The fration of group member who has the same region (country, province, city) as u.
Local Structure	Number of friends already in the group.
	Number of friend in group C who are classified as acitive inviters.
	Number of connected components formed by friends already in the group.



(a) Example of local structure (b) #Friend in group (c) Structure diversity

Figure 4.8: Local structure pattern of invitations. (a): Illustration of a potential invitee's ego networks. Give a group C, blue nodes represent user v's friends who have been in the group, while the white nodes denote those users in v's ego networks who did not join the group. (b): The probability p of being invited to a WeChat group as a function of the number of friends k already in the group. Error bars represent 95% confidence interval. (c): The effect of structure diversity. Structural diversity is represented by the number of connected components formed by the friends already in the group. Horizontal axis is the number of connected components formed by friends already in the group and the vertical axis is the probability p of being invited to a WeChat group.

4.5.2 Membership Cascade Prediction

Now we have seen how the membership cascade process can be affected by both users' behavioral pattern as well as the local structure surrounding a user. In this subsection, we propose a prediction model by integrating a comprehensive set of behavioral and structural features. This can be used in practice to make effective inference on the membership cascade process. Specifically, given the historical behavior of group members as well as the local social structure, can we predict which members in the group are more likely to be active and invite new users to the group chat and to whom will he/she send invitations to?

To address the issue, we separately model the inviter prediction and invitee prediction problems.

Inviter Prediction. At some timestamp T, for a given group C and a specified user $u \in C$, our learning task here is to predict whether u will become active and invite friends to C during the time interval $(T, T + \Delta t]$.

Invitee Prediction. At some time T, for a given group C and a user $u \in fringe(C)$, our learning task here is to predict whether u (one-hop neighbor of current members) will be invited to C during the time interval $(T, T + \Delta t]$.

For both prediction task, we construct each training example by randomly selecting T from 10 minutes to 1 month, and fixing Δt = 1 day. We also notice that, on average, only 5.6% of group members have invitation action and fewer than 1% among users in the fringe of groups are invited, causing the number of positive examples and negative examples quite unbalanced. We thus down-sample [105] the size of negative examples and maintain a positive/negative ratio of 1:2 in our training set.

Table 4.7: Performance of inviter/invitee prediction and inviter/invitee-level feature contribution analysis (%).

Task	Features used	AUC	Prec.	Rec.	F1
Inviter	All	**95.31**	**85.95**	**88.39**	**87.15**
	-History Behavior	91.52	82.07	84.31	83.17
	-Local Structure	93.22	84.50	87.04	85.75
Invitee	All	**98.66**	**54.55**	**93.47**	**68.89**
	-Demographics	98.05	45.76	94.68	61.70
	-Local Structure	89.29	11.85	76.53	20.52

We incorporate both group-level and inviter-level features seen in Table 4.5 and Table 4.6 for training the inviter model; and use instead the group-level and invitee-level features for invitee model. We further train the dataset using

support vector machine (SVM) [54] with 10-fold cross validation. The prediction performance results are shown in Table 4.7. We see that our model is quite effective, with AUC of 95.31% in predicting inviter, and an AUC of 98.66% in predicting invitee. We further investigated how each set of features affects the training performance by considering only one at a time. Quite interestingly, we find that historical behavioral features can be important factors in the task of predicting inviter; while local structural features are the dominant ones in predicting invitee. The information of demographics exert little affect on the performance of predicting invitee, which implies that our model can be generalizable without requiring user-specific attributes.

4.6 Conclusion

Summary. In this paper, we studied the formation and evolution of chat groups in the context of social messaging — their lifecycles, the change in their underlying structures over time, and the diffusion processes by which they develop new members. We use a large collection of anonymized data from WeChat group messaging platform, providing analysis on dynamics of millions of groups by keeping track of their emergence, growth and demise over time. We discovered a strong dichotomy of groups existed in terms of their lifecycle, and defined two types of groups accordingly: long-term and short-term groups. First, we developed an effective separability model by taking into account a broad range of group-level features, showing that long-term and short-term groups are inherently distinct. We also found that the lifecycle of messaging groups is largely dependent on their social roles and functionalities in users' daily social experiences and specific purposes. Specifically, event-driven groups in gen-

eral have a shorter lifespan as apposed to those friendship groups serving for frequent catching-up purpose. Given the strong separability between the long-term and short-term groups, we further addressed the problem of early prediction in group longevity, and demonstrated that strong prediction results can be obtained even with a group's history up to one day.

In addition to modeling the growth and evolution from a group-level perspective, we also investigated the individual-level attributes of group members and study the diffusion process by which groups gain new members. We developed a membership cascade process model in which we consider users historical engagement behavior as well as the local social network structure that users embedded in. We demonstrated the effectiveness by achieving AUC of 95.31% in the inviter prediction model using all features, and an AUC of 98.66% in the invitee prediction model.

Future Research. Ours findings raise many important open questions that would be interesting to take into account in future research. First, our design of membership cascade model can be used in practice for group member recommendation, and may be potentially integrated into current WeChat platform. This can further motivate research on conducting online experiments and investigating whether users are likely to adopt the group member recommendations, and under what circumstances. Such studies will also lead to design of better group chat platforms and engage users more effectively in the long run.

Part II

Machine Vision for Understanding Visual Contents on the Web

CHAPTER 5

VISUALIZING AND UNDERSTANDING DEEP NEURAL NETWORKS

This section is written in collaboration with Jason Yosinski, Jeff Clune and John Hopcroft. The work was published in Proceedings of the 4th International Conference on Learning Representation in 2016, and Journal in Machine Learning Research.

5.1 Introduction

Many recent studies have focused on understanding deep neural networks from both a theoretical perspective [16, 137, 133, 147, 63] and from an empirical perspective [52, 51, 174, 160, 197, 140, 194, 126, 195, 198]. In this paper we continue this trajectory toward attaining a deeper understanding of neural net training by proposing a new approach. We begin by noting that modern deep neural networks (DNNs) exhibit an interesting phenomenon: networks trained starting at different random initializations frequently converge to solutions with similar performance (see [43] and Section 5.2 below). Such similar performance by different networks raises the question of to what extent the learned internal representations differ: Do the networks learn radically different sets of features that happen to perform similarly, or do they exhibit *convergent learning*, meaning that their learned feature representations are largely the same? This paper makes a first attempt at asking and answering these questions. Any improved understanding of what neural networks learn should improve our ability to design better architectures, learning algorithms, and hyperparameters, ultimately enabling more capable models. For instance, distributed data-parallel neural network training is more complicated than distributed data-parallel training of

convex models because periodic direct averaging of model parameters is not an effective strategy: perhaps solving a neuron correspondence problem before averaging would mitigate the need for constant synchronization. As another example, if networks converge to diverse solutions, then perhaps additional performance improvements are possible via training multiple models and then using model compilation techniques to realize the resulting ensemble in a single model.

In this paper, we investigate the similarities and differences between the representations learned by neural networks with the same architecture trained from different random initializations. We employ an architecture derived from AlexNet [100] and train multiple networks on the ImageNet dataset [46] (details in Section 5.2). We then compare the representations learned across different networks. We demonstrate the effectiveness of this method by both visually and quantitatively showing that the features learned by some neuron clusters in one network can be quite similar to those learned by neuron clusters in an independently trained neural network. Our specific contributions are asking and shedding light on the following questions:

1. By defining a measure of similarity between units[1] in different neural networks, can we come up with a permutation for the units of one network to bring it into a one-to-one alignment with the units of another network trained on the same task? Is this matching or alignment close, because features learned by one network are learned nearly identically somewhere on the same layer of the second network, or is the approach ill-fated, because the representations of each network are unique? (Answer: a core repre-

[1]Note that we use the words "filters", "channels", "neurons", and "units" interchangeably to mean channels for a convolutional layer or individual units in a fully connected layer.

sentation is shared, but some rare features are learned in one network but not another; see Section 5.3).

2. Are the above one-to-one alignment results robust with respect to different measures of neuron similarity? (Answer: yes, under both linear correlation and estimated mutual information metrics; see Section 5.3.2).
3. To the extent that an accurate one-to-one neuron alignment is not possible, is it simply because one network's representation space is a rotated version[2] of another's? If so, can we find and characterize these rotations? (Answers: by learning a sparse weight LASSO model to predict one representation from only a few units of the other, we can see that the transform from one space to the other can be possibly decoupled into transforms between small subspaces; see Section 5.4).
4. Can we further cluster groups of neurons from one network with a similar group from another network? (Answer: yes. To approximately match clusters, we adopt a spectral clustering algorithm that enables many-to-many mappings to be found between networks. See Section A.3).
5. For two neurons detecting similar patterns, are the activation statistics similar as well? (Answer: mostly, but with some differences; see Section A.4).

5.2 Experimental Setup

All networks in this study follow the basic architecture laid out by [100], with parameters learned in five convolutional layers (`conv1` – `conv5`) followed by

[2]Or, more generally, a space that is an affine transformation of the first network's representation space.

three fully connected layers (fc6 – fc8). The structure is modified slightly in two ways. First, [100] employed limited connectivity between certain pairs of layers to enable splitting the model across two GPUs.[3] Here we remove this artificial group structure and allow all channels on each layer to connect to all channels on the preceding layer, as we wish to study only the group structure, if any, that arises naturally, not that which is created by architectural choices. Second, we place the local response normalization layers after the pooling layers following the defaults released with the Caffe framework, which does not significantly impact performance [82]. Networks are trained using Caffe on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 dataset [46]. Further details and the complete code necessary to reproduce these experiments is available at `https://github.com/yixuanli/convergent_learning`.

We trained four networks in the above manner using four different random initializations. We refer to these as Net1, Net2, Net3, and Net4. The four networks perform very similarly on the validation set, achieving top-1 accuracies of 58.65%, 58.73%, 58.79%, and 58.84%, which are similar to the top-1 performance of 59.3% reported in the original study [100].

We then aggregate certain statistics of the activations within the networks. Given a network Net$n$ trained in this manner, the scalar random variable $X_{l,i}^{(n)}$ denotes the series of activation values produced over the entire ILSVRC validation dataset by unit i on layer $l \in \{\text{conv1}, \text{conv2}, \text{conv3}, \text{conv4}, \text{conv5}, \text{fc6}, \text{fc7}\}$.[4] We

[3] In [100] the conv2, conv4, and conv5 layers were only connected to half of the preceding layer's channels.

[4] For the fully connected layers, the random variable $X_{l,i}^{(n)}$ has one specific value for each input image; for the convolutional layers, the value of $X_{l,i}^{(n)}$ takes on different values at each spatial position. In other words, to sample an $X_{l,i}^{(n)}$ for an FC layer, we pick a random image from the validation set; to sample $X_{l,i}^{(n)}$ for a conv layer, we sample a random image and a random position within the conv layer.

collect the following statistics by aggregating over the validation set (and in the case of convolutional layers also over spatial positions):

$$
\begin{aligned}
\text{Mean:} \quad & \mu_{l,i}^{(n)} = \mathbb{E}[X_{l,i}^{(n)}] \\
\text{Standard deviation:} \quad & \sigma_{l,i}^{(n)} = \sqrt{(\mathbb{E}[(X_{l,i}^{(n)} - \mu_{l,i}^{(n)})^2])} \\
\text{Within-net correlation:} \quad & c_{l,i,j}^{(n)} = \mathbb{E}[(X_{l,i}^{(n)} - \mu_{l,i}^{(n)})(X_{l,j}^{(n)} - \mu_{l,j}^{(n)})]/\sigma_{l,i}^{(n)}\sigma_{l,j}^{(n)} \\
\text{Between-net correlation:} \quad & c_{l,i,j}^{(n,m)} = \mathbb{E}[(X_{l,i}^{(n)} - \mu_{l,i}^{(n)})(X_{l,j}^{(m)} - \mu_{l,j}^{(m)})]/\sigma_{l,i}^{(n)}\sigma_{l,j}^{(m)}
\end{aligned}
$$

Intuitively, we compute the mean and standard deviation of the activation of each unit in the network over the validation set. For convolutional layers, we compute the mean and standard deviation of each channel. The mean and standard deviation for a given network and layer is a vector with length equal to the number of channels (for convolutional layers) or units (for fully connected layers).[5] The within-net correlation values for each layer can be considered as a symmetric square matrix with side length equal to the number of units in that layer (e.g. a 96×96 matrix for conv1 as in Figure 5.1a,b). For a pair of networks, the between-net correlation values also form a square matrix, which in this case is not symmetric (Figure 5.1c,d).

We use these correlation values as a way of measuring how related the activations of one unit are to another unit, either within the network or between networks. We use correlation to measure similarity because it is independent of the scale of the activations of units. Within-net correlation quantifies the similarity between two neurons in the same network; whereas the between-net correlation matrix quantifies the similarity of two neurons from different neural networks.

[5] For reference, the number of channels for conv1 to fc8 is given by: $\mathcal{S} = \{96, 256, 384, 384, 256, 4096, 4096, 1000\}$. The corresponding size of the correlation matrix in each layer is: $\{s^2 \mid \forall s \in \mathcal{S}\}$. Furthermore, the spatial extents of each channel in each convolutional layer is given by: $\{\text{conv1} : 55 \times 55, \text{conv2} : 27 \times 27, \text{conv3} : 13 \times 13, \text{conv4} : 13 \times 13, \text{conv5} : 13 \times 13\}$

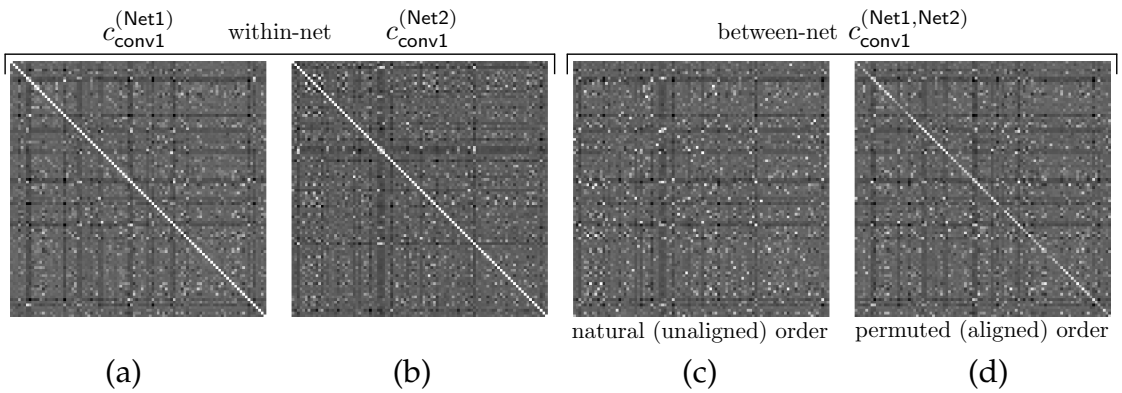


Figure 5.1: Correlation matrices for the conv1 layer, displayed as images with minimum value at black and maximum at white. **(a,b)** Within-net correlation matrices for Net1 and Net2, respectively. **(c)** Between-net correlation for Net1 vs. Net2. **(d)** Between-net correlation for Net1 vs. a version of Net2 that has been permuted to approximate Net1's feature order. The partially white diagonal of this final matrix shows the extent to which the alignment is successful; see Figure 5.3 for a plot of the values along this diagonal and further discussion.

Note that the units compared are always on the same layer on the network; we do not compare units between different layers. In the Supplementary Information (see Figure A.1), we plot the activation values for several example high correlation and low correlation pairs of units from conv1 and conv2 layers; the simplicity of the distribution of values suggests that the correlation measurement is an adequate indicator of the similarity between two neurons. To further confirm this suspicion, we also tested with a full estimate of the mutual information between units and found it to yield similar results to correlation (see Section 5.3.2).

5.3 One-to-One Alignment Between Features Learned by Different Neural Networks

We would like to investigate the similarities and differences between multiple training runs of same network architecture. Due to symmetries in the architecture and weight initialization procedures, for any given parameter vector that is found, one could create many equivalent solutions simply by permuting the unit orders within a layer (and permuting the outgoing weights accordingly). Thus, as a first step toward analyzing the similarities and differences between different networks, we ask the following question: if we allow ourselves to permute the units of one network, to what extent can we bring it into alignment with another? To do so requires finding equivalent or nearly-equivalent units across networks, and for this task we adopt the magnitude independent measures of correlation and mutual information. We primarily give results with the simpler, computationally faster correlation measure (Section 5.3.1), but then confirm the mutual information measure provides qualitatively similar results (Section 5.3.2).

5.3.1 Alignment via Correlation

As discussed in Section 5.2, we compute within-net and between-net unit correlations. Figure 5.1 shows the within-net correlation values computed between units on a network and other units on the same network (panels a,b) as well as the between-net correlations between two different networks (panel c). We find matching units between a pair of networks — here Net1 and Net2 — in

two ways. In the first approach, for each unit in Net1, we find the unit in Net2 with maximum correlation to it, which is the max along each row of Figure 5.1c. This type of assignment is known as a bipartite *semi-matching* in graph theory [111], and we adopt the same nomenclature here. This procedure can result in multiple units of Net1 being paired with the same unit in Net2. Figure 5.2 shows the eight highest correlation matched features and eight lowest correlation matched features using the semi-matching approach (corresponding to the leftmost eight and rightmost eight points in Figure 5.3). To visualize the functionality each unit, we plot the image patch from the validation set that causes the highest activation for that unit. For all the layers shown, the most correlated filters (on the left) reveal that there are nearly perfect counterparts in each network, whereas the low-correlation filters (on the right) reveal that there are many features learned by one network that are unique and thus have no corollary in the other network.

An alternative approach is to find the one-to-one assignment between units in Net1 and Net2 without replacement, such that every unit in each network is paired with a unique unit in the other network. This more common approach is known as bipartite *matching*.[6] A matching that maximizes the sum of the chosen correlation values may be found efficiently via the Hopcroft-Karp algorithm [**?**] after turning the between-net correlation matrix into a weighted bipartite graph. Figure 5.1c shows an example between-net correlation matrix; the max weighted matching can be thought of as a path through the matrix such that each row and each column are selected exactly once, and the sum of elements along the path is maximized. Once such a path is found, we can permute the

[6]Note that the *semi-matching* is "row-wise greedy" and will always have equal or better sum of correlation than the *matching*, which maximizes the same objective but must also satisfy global constraints.

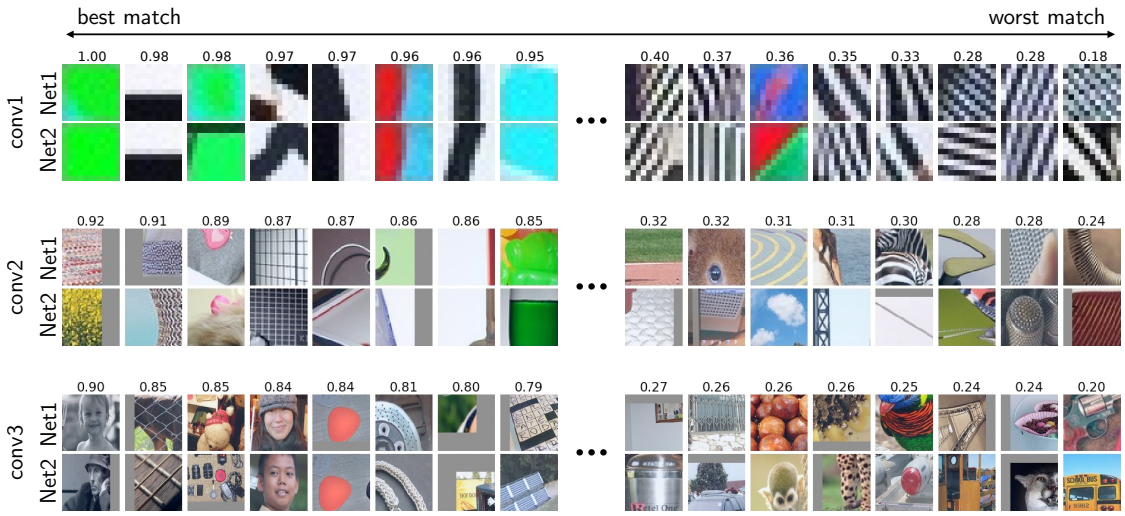


Figure 5.2: With assignments chosen by semi-matching, the eight best (highest correlation, left) and eight worst (lowest correlation, right) matched features between Net1 and Net2 for the conv1 – conv3 layers. For all layers visualized, (1) the most correlated filters are near perfect matches, showing that many similar features are learned by independently trained neural networks, and (2) the least correlated features show that many features are learned by one network and are not learned by the other network, at least not by a single neuron in the other network. The results for the conv4 and conv5 layers can be found in the Supplementary Material (see Figure A.2).

units of Net2 to bring it into the best possible alignment with Net1, so that the first channel of Net2 approximately matches (has high correlation with) the first channel of Net1, the second channels of each also approximately match, and so on. The correlation matrix of Net1 with the permuted version of Net2 is shown in Figure 5.1d. Whereas the diagonal of the self correlation matrices are exactly one, the diagonal of the permuted between-net correlation matrix contains values that are generally less than one. Note that the diagonal of the permuted between-net correlation matrix is bright (close to white) in many places, which shows that for many units in Net1 it is possible to find a unique, highly corre-

lated unit in Net2.

Figure 5.3 shows a comparison of assignments produced by the semi-matching and matching methods for the conv1 layer (Figure A.3 shows results for other layers). Insights into the differing representations learned can be gained from both assignment methods. The first conclusion is that for most units, particularly those with the higher semi-matching and matching correlations (Figure 5.3, left), the semi-matching and matching assignments coincide, revealing that for many units a one-to-one assignment is possible. Both methods reveal that the average correlation for one-to-one alignments varies from layer to layer (Figure 5.4), with the highest matches in the conv1 and conv5 layers, but worse matches in between. This pattern implies that the path from a relatively matchable conv1 representation to conv5 representation passes through an intermediate middle region where matching is more difficult, suggesting that what is learned by different networks on conv1 and conv2 is more convergent than conv3, conv4 and conv5. This result may be related to previously observed greater complexity in the intermediate layers as measured through the lens of optimization difficulty [194].

Next, we can see that where the semi-matching and matching differ, the matching is often much worse.One hypothesis for why this occurs is that the two networks learn different numbers of units to span certain subspaces. For example, Net1 might learn a representation that uses six filters to span a subspace of human faces, but Net2 learns to span the same subspace with five filters. With unique matching, five out of the six filters from Net1 may be matched to their nearest counterpart in Net2, but the sixth Net1 unit will be left without a counterpart and will end up paired with an almost unrelated filter.

Finally, with reference to Figure 5.3 (but similarly observable in Figure A.3 for other layers), another salient observation is that the correlation of the semi-matching falls significantly from the best-matched unit (correlations near 1) to the lowest-matched (correlations near 0.3). This indicates that some filters in Net1 can be paired up with filters in Net2 with high correlation, but other filters in Net1 and Net2 are network-specific and have no high-correlation pairing in the alternate network, implying that those filters are rare and not always learned. This holds across the conv1 – conv5 layers.

5.3.2 Alignment via Mutual Information

Because correlation is a relatively simple mathematical metric that may miss some forms of statistical dependence, we also performed one-to-one alignments of neurons by measuring the *mutual information* between them. Mutual information measures how much knowledge one gains about one variable by knowing the value of another. Formally, the mutual information of the two random variables $X_{l,i}^{(n)}$and $X_{l,j}^{(m)}$ representing the activation of the i-th neuron in Net$n$ and the j-th neuron in Net$m$, is defined as:

$$I\left(X_{l,i}^{(n)}; X_{l,j}^{(m)}\right) = \sum_{a \in X_{l,i}^{(n)}} \sum_{b \in X_{l,j}^{(m)}} p(a,b) \log\left(\frac{p(a,b)}{p(a)p(b)}\right),$$

where $p(a,b)$ is the joint probability distribution of $X_{l,i}^{(n)}$ and $X_{l,j}^{(m)}$, and $p(a)$ and $p(b)$ are their marginal probability distributions, respectively. The within-net mutual information matrix and between-net mutual information matrix have the same shapes as their equivalent correlation matrices.

We apply the same matching technique described in Section 5.3.1 to the

between-net mutual information matrix,[7] and compare the highest and lowest mutual information matches (Figure A.4) to those obtained via correlation (Figure 5.2). The results are qualitatively the same. For example, seven out of eight best matched pairs in the conv1 layer stay the same. These results suggest that correlation is an adequate measurement of the similarity between two neurons, and that switching to a mutual information metric would not qualitatively change the correlation-based conclusions presented above.

5.4 Relaxing the One-to-One Constraint to Find Sparse, Few-to-One Mappings

The preceding section showed that, while some neurons have a one-to-one match in another network, for other neurons no one-to-one match exists (with correlation above some modest threshold). For example, 17% of conv1 neurons in Net1 have no match in Net2 with a correlation above 0.5 (Figure 5.3); this number rises to 37% for conv2, 63% for conv3, and 92% for conv4, before dropping to 75% for conv5 (see Figure A.3).

These numbers indicate that, particularly for intermediate layers, a simple one-to-one mapping is not a sufficient model to predict the activations of some neurons in one network given the activations of neurons in another network (even with the same architecture trained on the same task). That result could either be because the representations are unique (i.e. not convergent), or because

[7] The mutual information between each pair of neurons is estimated using 1D and 2D histograms of paired activation values over 60,000 random activation samples. We discretize the activation value distribution into percentile bins along each dimension, each of which captures 5% of the marginal distribution mass. We also add a special bin with range $(-\inf, 10^{-6}]$ in order to capture the significant mass around 0.

the best possible one-to-one mapping is insufficient to tell the complete story of how one representation is related to another. We can think of a one-to-one mapping as a model that predicts activations in the second network by multiplying the activations of the first by a permutation matrix — a square matrix constrained such that each row and each column contain a single one and the rest zeros. Can we do better if we learn a model without this constraint?

We can relax this one-to-one constraint to various degrees by learning a *mapping layer* with an L1 penalty (known as a LASSO model, [175]), where stronger penalties will lead to sparser (more few-to-one or one-to-one) mappings. This sparsity pressure can be varied from quite strong (encouraging a mostly one-to-one mapping) all the way to zero, which encourages the learned linear model to be dense. More specifically, to predict one layer's representation from another, we learn a single mapping layer from one to the other (similar to the "stitching layer" in [117]). In the case of convolutional layers, this mapping layer is a convolutional layer with 1×1 kernel size and number of output channels equal to the number of input channels. The mapping layer's parameters can be considered as a square weight matrix with side length equal to the number of units in the layer; the layer learns to predict any unit in one network via a linear weighted sum of any number of units in the other. The model and resulting square weight matrices are shown in Figure 5.5. This layer is then trained to minimize the sum of squared prediction errors plus an L1 penalty, the strength of which is varied.[8]

[8]Both representations (input and target) are taken after the relu is applied. Before training, each channel is normalized to have mean zero and standard deviation $1/\sqrt{N}$, where N is the number of dimensions of the representation at that layer (e.g. $N = 55 \cdot 55 \cdot 96 = 290400$ for conv1). This normalization has two effects. First, the channels in a layer are all given equal importance, without which the channels with large activation values (see Figure A.9) dominate the cost and are predicted well at the expense of less active channels, a solution which provides little information about the less active channels. Second, the representation at any layer for a single image becomes approximately unit length, making the initial cost about the same on all

Mapping layers are trained for layers conv1 – conv5. The average squared prediction errors for each are shown in Table 5.1 for a variety of L1 penalty weights (i.e. different decay values). For the conv1 and conv2 layers, the prediction errors do not rise with the imposition of a sparsity penalty until a penalty greater than 10^{-3}. A sparsity penalty as high as 10^{-3} results in mapping layer models that are nearly as accurate as their dense counterparts, but that contain mostly zero weights. Additional experiments for conv1 revealed that a penalty multiplier of $10^{-2.6}$ provides a good trade-off between sparsity and accuracy, resulting in a model with sparse prediction loss 0.235, and an average of 4.7 units used to predict each unit in the target network (i.e. an average of 4.7 significantly non-zero weights). For conv2 the 10^{-3} multiplier worked well, producing a model with an average of 2.5 non-zero connections per predicted unit. The mapping layers for higher layers (conv3 – conv5) showed poor performance even without regularization, for reasons we do not yet fully understand, so further results on those layers are not included here. Future investigation with different hyperparameters or different architectures (e.g. multiple hidden layers) could train more powerful predictive models for these layers.

The one-to-one results of Section 5.3 considered in combination with these results on sparse prediction shed light on the open, long-standing debate about the extent to which learned representations are local vs. distributed : The units that match well one-to-one suggest the presence of a *local* code, where each of these dimensions is important enough, independent enough, or privileged enough in some other way to be relearned by different networks. Units that do not match well one-to-one, but are predicted well by a sparse model, suggest

layers and allowing the same learning rate and SGD momentum hyperparameters to be used for all layers. It also makes the effect of specific L1 multipliers approximately commensurate and allows for rough comparison of prediction performance between layers, because the scale is constant.

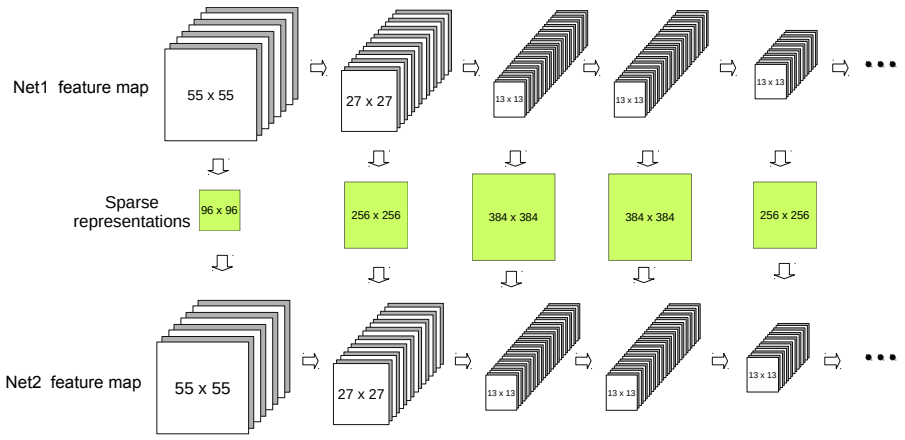


Figure 5.5: A visualization of the network-to-network sparse "mapping layers" (green squares). The layers are trained independently of each other and with an L1 weight penalty to encourage sparse weights.

the presence, along those dimensions, of *slightly distributed* codes.

The results could have been otherwise: if all units could accurately be matched one-to-one, we would suspect a local code across the whole layer. On the other hand, if making predictions from one network to another required a dense affine transformation, then we would interpret the code as *fully distributed*, with each unit serving only as a basis vector used to span a large dimensional subspace, whose only requirement was to have large projection onto the subspace (to be useful) and small projection onto other basis vectors (to be orthogonal). The story that actually emerges is that the first two layers use a mix of a local and a slightly distributed code.

Figure 5.6 shows a visualization of the learned weight matrix for conv1, along with a permuted weight matrix that aligns units from Net2 with the Net1 units that most predict them. We also show two example units in Net2 and, for each, the only three units in Net1 that are needed to predict their activation values.

These sparse prediction results suggest that small groups of units in each network span similar subspaces, but we have not yet identified or visualized the particular subspaces that are spanned. Below we present one approach to do so by using the sparse prediction matrix directly, and in the Supplementary Section A.3 we discuss a related approach using spectral clustering.

For a layer with s channels we begin by creating a $2s \times 2s$ block matrix B by concatenating the blocks $[I,\ W;\ W^T,\ I]$ where I is the $s \times s$ identity matrix and W is the learned weight matrix. Then we use Hierarchical Agglomerative Clustering (HAC), as implemented in Scikit-learn [148] to recursively cluster individual units into clusters, and those clusters into larger clusters, until all have been joined into one cluster. The HAC algorithm as adapted to this application works in the following way: (1) For all pairs of units, we find the biggest off-diagonal value in B, i.e. the largest prediction weight; (2) We pair those two units together into a cluster and consider it as a single entity for the remainder of the algorithm; (3) We start again from step 2 using the same process (still looking for the biggest value), but whenever we need to compare unit $\leftrightarrow$ cluster or cluster $\leftrightarrow$ cluster, we use the average unit $\leftrightarrow$ unit weight over all cross-cluster pairs;(4) Eventually the process terminates when there is a single cluster.[9]

The resulting clustering can be interpreted as a tree with units as leaves, clusters as intermediate nodes, and the single final cluster as the root node. In Figure 5.7 we plot the B matrix with rows and columns permuted together in the order leaves are encountered when traversing the tree, and intermediate nodes are overlaid as lines joining their subordinate units or clusters. For clarity, we color the diagonal pixels (which all have value one) with green or red if the associated unit came from Net1 or Net2, respectively.

[9]For example, in the conv1 layer with 96 channels, this happens after $96 \cdot 2 - 1 = 191$ steps.

Plotted in this way, structure is readily visible: Most parents of leaf clusters (the smallest merges shown as two blue lines of length two covering a 2×2 region) contain one unit from Net1 and one from Net2. These units can be considered most predictive of each other.[10] Slightly higher level clusters show small subspaces, comprised of multiple units from each network, where multiple units from one network are useful for predicting activations from the other network (see the example zoomed regions on the right side of Figure 5.7).

The HAC method employs greedy merges, which could in some cases be suboptimal. In the Supplementary Section A.3 we explore a related method that is less greedy, but operates on the denser correlation matrix instead. Future work investigating or developing other methods for analyzing the structure of the sparse prediction matrices may shed further light on the shared, learned subspaces of independently trained networks.

5.5 Conclusions

We have demonstrated a method for quantifying the feature similarity between different, independently trained deep neural networks. We show how insights may be gain by approximately aligning different neural networks on a feature or subspace level by blending three approaches: a bipartite matching that makes one-to-one assignments between neurons, a sparse prediction and clustering approach that finds one-to-many mappings, and a spectral clustering approach that finds many-to-many mappings. Our main findings include:

[10] Note that the upper right corner of B is $W = W_{1\to2}$, the matrix predicting Net1 $\to$ Net2, and the lower left is just the transpose $W_{1\to2}^T$. The corners could instead be $W_{1\to2}$ and $W_{2\to1}$, respectively.

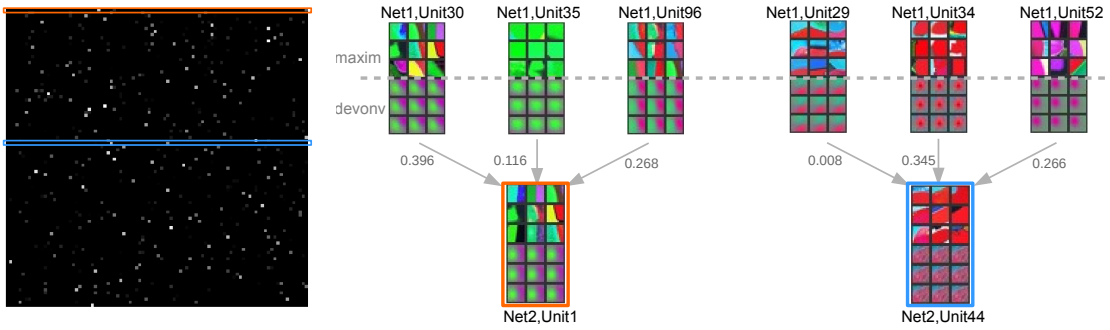


Figure 5.6: **(left)** The learned mapping layer from Net1 to Net2 for the conv1 layer. **(right)** Two example units (bottom) in Net2 — which correspond to the same colored rows in the left weight matrix — together with, for each, the only three units in Net1 that are needed to predict their activation. To fully visualize the functionality each unit, we plot the top 9 image patches from the validation set that causes the highest activation for that unit ("maxim"), as well as its corresponding "deconv" visualization introduced by [197]. We also show the actual weight associated with each unit in Net1 in the sparse prediction matrix.

1. Some features are learned reliably in multiple networks, yet other features are not consistently learned.

2. Units learn to span low-dimensional subspaces and, while these subspaces are common to multiple networks, the specific basis vectors learned are not.

3. The representation codes are a mix between a local (single unit) code and slightly, but not fully, distributed codes across multiple units.

4. The average activation values of neurons vary considerably within a network, yet the mean activation values across different networks converge to an almost identical distribution.

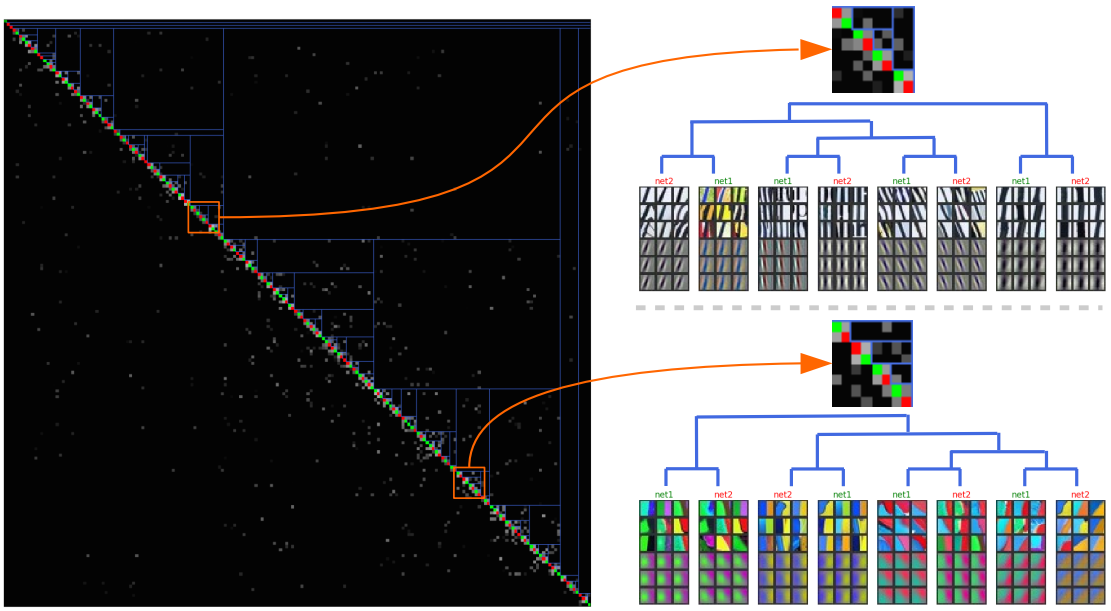


Figure 5.7: The results of the Hierarchical Agglomerative Clustering (HAC) algorithm described in Section 5.4 on the conv1 layer. Left: The B matrix permuted by the tree-traversal order of leaf nodes. Pixels on the diagonal are leaf nodes and represent original units of either network (green for Net1 and red for Net2). The brighter the gray pixel is, the larger the weight is in the matrix. See text for a complete interpretation. Right: Two zoomed in regions of the diagonal, showing two different four-dimensional subspaces spanned by four units in each network. The top 9 and bottom 9 images correspond to the maxim and deconv visualizations, respectively. Best viewed digitally with zoom.

5.6 Future Work

The findings in this paper open up new future research directions, for example: (1) Model compression. How would removing low-correlation, rare filters affect performance? (2) Optimizing ensemble formation. The results show some features (and subspaces) are shared between independently trained DNNs, and some are not. This suggests testing how feature correlation among different

DNNs in an ensemble affects ensemble performance. For example, the "shared cores" of multiple networks could be deduplicated, but the unique features in the tails of their feature sets could be kept. (3) Similarly, one could (a) post-hoc assemble ensembles with greater diversity, or even (b) directly encourage ensemble feature diversity during training. (4) Certain visualization techniques, e.g., deconv [197], DeepVis [195], have revealed neurons with multiple functions (e.g. detectors that fire for wheels and faces). The proposed matching methods could reveal more about why these arise. Are these units consistently learned because they are helpful or are they just noisy, imperfect features found in local optima? (5) Model combination: can multiple models be combined by concatenating their features, deleting those with high overlap, and then fine-tuning? (6) Apply the analysis to networks with different architectures — for example, networks with different numbers of layers or different layer sizes — or networks trained on different subsets of the training data. (7) Study the correlations of features in the same network, but across training iterations, which could show whether some features are trained early and not changed much later, versus perhaps others being changed in the later stages of fine-tuning. This could lead to complementary insights on learning dynamics to those reported by [52]. (8) Study whether particular regularization or optimization strategies (e.g., dropout, ordered dropout, path SGD, etc.) increase or decrease the convergent properties of the representations to facilitate different goals (more convergent would be better for data-parallel training, and less convergent would be better for ensemble formation and compilation).

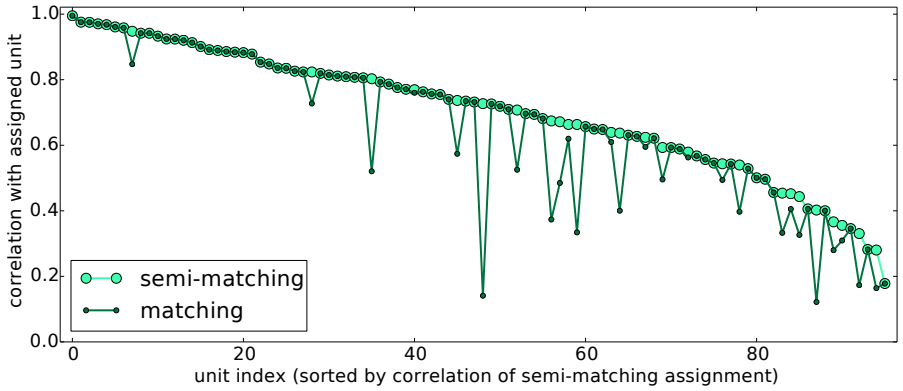


Figure 5.3: Correlations between paired conv1 units in Net1 and Net2. Pairings are made via semi-matching (light green), which allows the same unit in Net2 to be matched with multiple units in Net1, or matching (dark green), which forces a unique Net2 neuron to be paired with each Net1 neuron. Units are sorted by their semi-matching values. See text for discussion.

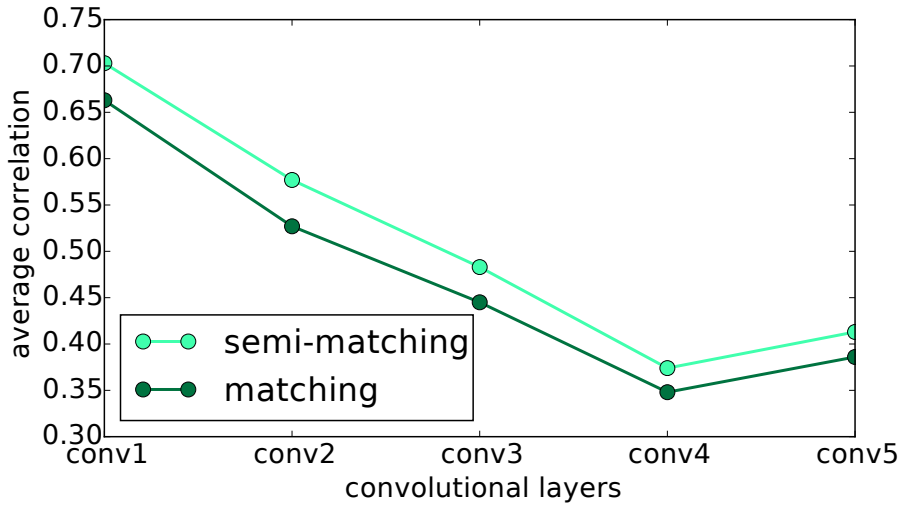


Figure 5.4: Average correlations between paired conv1 units in Net1 and Net2. Both semi-matching (light green) and matching (dark green) methods suggest that features learned in different networks are most convergent on conv1 and least convergent on conv4.

	Sparse Prediction Loss (after 4,500 iterations)					
	decay 0	decay 10^{-5}	decay 10^{-4}	decay 10^{-3}	decay 10^{-2}	decay 10^{-1}
conv1	**0.170**	**0.169**	**0.162**	**0.172**	0.484	0.517
conv2	**0.372**	**0.368**	**0.337**	**0.392**	0.518	0.514
conv3	0.434	0.427	0.383	0.462	0.497	0.496
conv4	0.478	0.470	0.423	0.477	0.489	0.488
conv5	0.484	0.478	0.439	0.436	0.478	0.477

Table 5.1: Average prediction error for mapping layers with varying L1 penalties (i.e. decay terms). Larger decay parameters enforce stronger sparsity in the learned weight matrix. Notably, on conv1 and conv2, the prediction errors do not rise much compared to the dense (decay = 0) case with the imposition of a sparsity penalty until after an L1 penalty weight of over 10^{-3} is used. This region of roughly constant performance despite increasing sparsity pressure is shown in bold. That such extreme sparsity does not hurt performance implies that each neuron in one network can be predicted by only one or a few neurons in another network. For the conv3, conv4, and conv5 layers, the overall error is higher, so it is difficult to draw any strong conclusions regarding those layers. The high errors could be because of the uniqueness of the learned representations, or the optimization could be learning a suboptimal mapping layer for other reasons.

CHAPTER 6

TRAINING NEURAL NETWORKS ENSEMBLES THROUGH COMMON SUBSPACE

This section is written in collaboration with Shuang Li, Matt Kusner, Karthik Sridharan, Kilian Weinberger and John Hopcroft.

6.1 Introduction

Averaging classifiers is an effective method to reduce model variance by approximating the expected classifier. Ensemble methods [28, 34, 67, 103, 199] leverage this fact to obtain improved generalization performance [97]. For example, the Random Forests [28] algorithm elevates the modest CART [29] algorithm to one of the most competitive classifiers within machine learning. Here, slightly modified CART trees are bagged [28], where each tree is trained on a different subset of the data (drawn uniformly with replacement) and splits are randomized. The additional randomization of the decision trees leads to high variance classifiers. This is advantageous as the ensemble size can be very large (in practice often exceeding 10,000 CART trees) due to the extremely efficient ID3 algorithm [150]. The large number of classifiers ensures that even in the presence of high variance the ensemble average approaches the expected model (which would have zero model variance).

In the wake of deep learning, ensembling is just as important as it has ever been. Nowadays most high profile competitions (e.g. Imagenet [47] or Kaggle[1])

[1] www.kaggle.com

are won by ensembles of deep learning architectures. As neural networks are often initialized with random weights, there appears to be a sufficient amount of natural variation that allows all networks to be trained on the entire data set.

Training deep networks is computationally expensive and can last for days or even weeks even on high performance hardware with GPU acceleration. Training ensembles of them increases the cost linearly and quickly becomes prohibitive for most researchers without access to industrial scale computational resources. Although the training of deep net ensembles can be trivially parallelized, few have access to sufficient GPU servers that can be deployed in parallel for long durations. As a result, ensembles of deep nets are typically small—averaging only a hand-full of classifiers. Consequently, the ensemble average still has high variance and does not approach the expected model.

In this paper we propose *Subspace Ensemble Networks* (SEN), a method that improves the generalization performance of small deep network ensembles. SEN trades off the variance of individual deep networks for increased model bias (along a carefully chosen direction). A key component to the success of deep learning is that neural networks automatically learn their own feature representation of the data. Similar to the standard ensemble approach of reducing the variance of the *output* by averaging the predictions, we also reduce the variance of the *internal representation* by aligning the learned features.

Neural networks that are trained on the same data set but with different initializations will learn partially correlated feature representations [122]. We reinforce this trend by decomposing the feature representation of the final layer into two components: (a) the common low-dimensional subspaces that are aligned across all ensemble members; and (b) the corresponding null spaces—

orthogonal to the subspaces—learn features specific to each particular neural network.

The aligned subspace distills the commonality between the individual networks' features and improves their respective feature quality but biases them towards the same representation. The null spaces allows each individual network to learn features that are unique to itself and make different generalization mistakes. The decomposition of representation space is therefore a natural way to trade-off the bias and variance of the ensemble members.

We evaluate SEN on six benchmark data sets and demonstrate that it yields significant performance gains over standard ensembles. In particular, it reduces the required ensemble sizes drastically. In 5 out of 6 data sets SEN with only 2 classifiers significantly outperforms ensembles of 20 deep nets.

6.2 Related Work and Preliminary

We first briefly review deep networks and our interpretation of them as generalized feature extractors. We then discuss model ensembling and the special case in which each model is a deep neural network. Throughout, we shall use lower case letters for scalars and functions, bold lower case for vectors, and upper case letters for matrices.

Deep Networks. We assume that we are provided with a labeled training set of n samples $\{(\boldsymbol{x}^1, \mathbf{y}^1), \ldots, (\boldsymbol{x}^n, \mathbf{y}^n)\} \sim P^n(\boldsymbol{x}, \mathbf{y})$, where $\boldsymbol{x}^i \in \mathbb{R}^d$ are the input features and $\mathbf{y}^i \in \Delta^k$ are the corresponding classification labels[2]. Deep networks

[2] Δ^k denotes the $k-1$ dimensional simplex of class probabilities.

[68, 157, 22] take an initial feature representation $\boldsymbol{x}$ and learn a nested sequence of representations that are designed to be more effective for classification. In general, we can view these approaches as generalized feature extractors that given an $\boldsymbol{x}$, produce high-level features $\boldsymbol{h} = \mathsf{Net}(\boldsymbol{x}) \in \mathbb{R}^p$, when passed through the deep network $\mathsf{Net}(\cdot)$. We jointly learn a linear classifier $W \in \mathbb{R}^{p \times k}$ to predict the class probability of each input. Specifically, we compute the linear classification $\mathbf{z} = W^\top \boldsymbol{h}$ and apply the softmax function $\sigma(\mathbf{z}) = \frac{e^{\mathbf{z}}}{\sum_{j=1}^k e^{z_j}}$ to estimate the corresponding class probability vector $\mathbf{y}$. We train our classifier W to minimize the cross-entropy loss $\ell(\cdot, \cdot) : \Delta^k \times \Delta^k \to \mathbb{R}_+$ between the true label (one-hot) distribution y and our estimated distribution $\sigma(\mathbf{z})$:

$$\ell(\mathbf{y}, \sigma(\mathbf{z})) = -\sum_{j=1}^{k} y_j \log \sigma(\mathbf{z})_j.$$

Ensemble Methods. In general an ensemble [28, 34, 48, 103, 199] is a collection of machine learning models that are averaged to improve their overall generalization performance by reducing the variance of the model space. The intuition is that because different models will tend to make different mistakes, by averaging the predictions of many models we can 'average over' these mistakes and thus improve classification [67]. This technique has notably been applied very successfully in Random Forests [28], which turn simple decision tree models into a highly-competitive classifier.

In this paper we will consider an ensemble of M *neural networks*, represented by M feature extractors $\mathsf{Net}_1, \ldots, \mathsf{Net}_M$. While it is possible to ensemble neural networks with different architectures, for simplicity, we assume that the M networks have the same architecture and further we denote the dimensionality of the last layer by D. Throughout this paper we shall use the shorthand nota-

tion $\boldsymbol{h}_i^{\alpha}$ to denote the vector $\mathsf{Net}_{\alpha}(\boldsymbol{x}^i)$. It will be understood that $\boldsymbol{h}_{\alpha}^{i}$ refers to the α^{th} network applied to the i^{th} sample. Figure 6.1 illustrates the architecture of a standard ensemble of (two) deep networks.

Structural Learning Our approach falls into a general category of structural learning. More closely related to our approach is the work by Ando and Zhang et al. [12, 13], who proposed the learning paradigm that allows for discovering a common predictive subspace shared by multiple *linear* models. This stream of research has demonstrated its success in semi-supervised learning [13], multi-task learning [142] and multi-label classification [81]. Blitzer et al. [26, 25] introduced a variant approach, structural correspondence learning (SCL) for domain adaptation. SCL enables learning a shared low-dimensional linear subspace from both the unlabeled source and target data. [24, 87] related structural learning and SCL to canonical correlation analysis (CCA) [75], a statistical method for discovering correlating basis vectors of two multivariate random variables. Our approach advances previous studies on structural learning by considering the setting in deep networks instead of linear models, and allows discovering the common low-dimensional predictive structure in the inner working of neural representations.

Neural Representation Alignment There is a growing body of work that investigates the representation alignment in multiple neural networks, which are either trained on the same task [122] or multi-modal data. Li et al. [122] found that features learned in multiple independently trained convolutional neural networks (CNNs) tend to span *similar* low-dimensional subspaces, although the basis vectors can be rotated in the subspaces. Ngiam et al. [139] described an au-

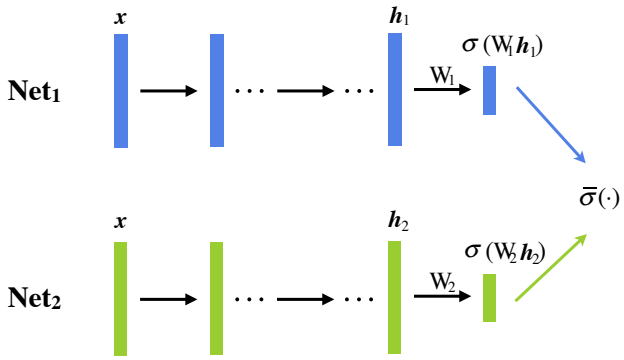


Figure 6.1: The paradigm of conventional ensemble neural networks. Models (here Net_1 and Net_2) are independently trained. The ensemble makes prediction by averaging over the individual models.

toencoder that learned audio-video representations through a shared bottleneck layer. Frome et al. [57] and Karpathy et al. [89, 88, 164] introduced models that learned to map images and words to a common semantic embedding. The success of visual-semantic alignment has shed light on important computer vision tasks such as visual question answering [14, 128] and image captioning [55, 189].

6.3 Subspace Ensemble Networks

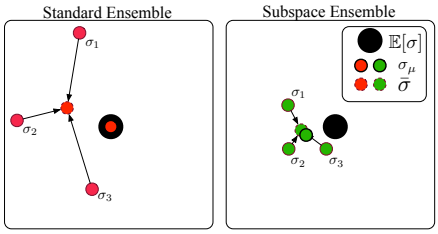


Figure 6.2: Illustration of SEN.

In general, we will not be able to ensemble more than 3 or 5 deep networks due to the massive computational cost of days or even weeks required to train a single network. In this section we describe an efficient technique to create an accurately biased small deep network ensemble, that in many cases outperforms much larger network ensembles.

Figure 6.2 illustrates the high level intuition behind our subspace ensemble method. In a standard ensemble (*left plot*), classifiers $\sigma_1, \ldots, \sigma_M$ are drawn from a distribution around the expected classifier $\sigma_\mu = \mathbb{E}[\sigma]$.

The ensemble approximates this expected classifier because $\bar{\sigma} \rightarrow \sigma_\mu$ as $M \rightarrow \infty$. If the classifier variance is high and the ensemble size M is small, this approximation will be poor. We therefore propose to sample *low variance* classifiers around a biased center $\sigma_\mu \neq \mathbb{E}[\sigma]$ however at low variance (*right plot*). We obtain the biased classifier σ_μ by enforcing a shared subspace across all networks. More precisely, the learned features $\boldsymbol{h}_\alpha$ is decomposed into two vectors within orthogonal sub-spaces $\boldsymbol{s}_\alpha$ and $\boldsymbol{s}_\alpha^\perp$. The first of the two shares a classifier V across all neural networks which all ensemble members to be close to a common "center classifier" σ_μ. The latter has classifier weights unique to each neural network and ensures a controlled amount of variance across the ensemble members. To facilitate this decomposition, we add an additional subspace decomposition layer to each neural network, as illustrated in Figure 6.3. In the following, we describe its individual components in detail.

Subspace Decomposition. We decompose the feature representation learned by Net_α into two orthogonal subspaces $\boldsymbol{s}_\alpha = \Theta \boldsymbol{h}_\alpha$ and $\boldsymbol{s}_\alpha^\perp = \hat{\Theta} \boldsymbol{h}_\alpha$. The two affine matrices are of dimensions $\Theta_\alpha \in \mathbb{R}^{d \times D}$ and $\hat{\Theta}_\alpha \in \mathbb{R}^{(D-d) \times D}$ and are constrained to have (approximately) orthonormal rows, *i.e.* precisely $\Theta\Theta^\top \approx I$ and $\hat{\Theta}\hat{\Theta}^\top \approx I$. The two sub-spaces are forced to be approximately orthogonal, *i.e.* $\Theta^\top \hat{\Theta} \approx 0$. Essentially, we divide the feature space into two different components. The first will be used for a shared, the second for a network specific classifier.

Subspace Classifiers. We define the first component of the feature repre-

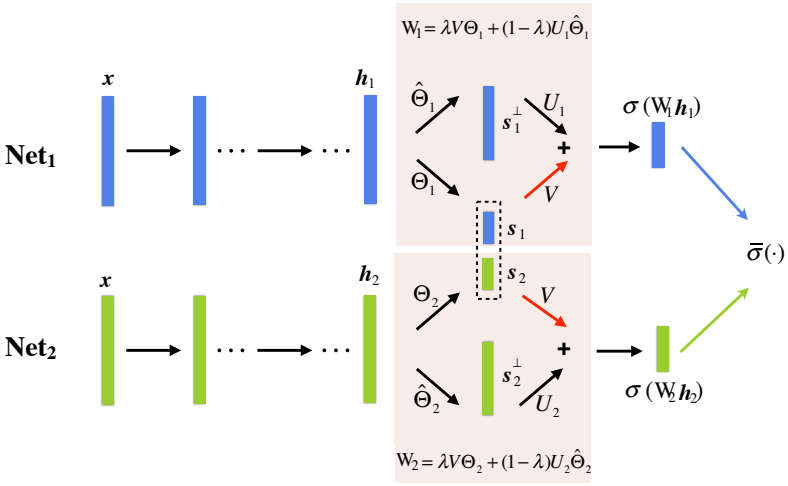


Figure 6.3: The Subspace Ensemble Network (SEN). The final hidden layer of each network is decomposed into two types of activations: 1. an aligned activation set $\boldsymbol{s}_1, \boldsymbol{s}_2$ and 2. a set of orthogonal activations $\boldsymbol{s}_1^{\perp}, \boldsymbol{s}_2^{\perp}$. The aligned activations capture generalized information about the classification task and the orthogonal activations explain the variance of the model class. The final output of each network is the softmax $\sigma(\cdot)$ over a weighted combination of the aligned and orthogonal activations.

sentation, $\boldsymbol{s}_\alpha$, as the *shared* space and the second component, $\boldsymbol{s}_\alpha^{\top}$, as the *network specific* space. For the shared space $\boldsymbol{s}$, we learn a *shared* weight matrix $V \in \mathbb{R}^{d \times k}$ which produces predictions $V\Theta_\alpha \boldsymbol{h}_\alpha$. Each row of V is a linear classifier for one of the k classes, applied and averaged over the d dimensions of the shared space. As these linear classifiers are shared across the ensemble members, the feature representation becomes aligned such that the transformed data points $\boldsymbol{s}_\alpha^i$ lie on the "right side" of the hyper-planes. In practice, the fully connected layer leading to the softmax can have hundreds or even thousands of dimensions, resulting in high-dimensional representation space. The dimensionality of the subspace d can be much smaller than the original feature dimensionality D. The shared subspace distills the most valuable predictive structure across the ensemble networks. It further increases the similarity across the networks and

benefits the classifier V, which is effectively trained on a much larger set of data points ($M \times n$) and therefore generalizes better to unseen test data.

Network Specific Null Space. In order for the ensemble compilation to work, it is important to allow classifiers to make independent mistakes (which is averaged out). It is therefore important to also have a feature representation and classifier weights that are unique to the individual model. We facilitate this requirement by maintaining a second, network specific weight matrix U_α for the null space $\boldsymbol{s}_\alpha^\top$. The weight matrix $U_\alpha \in \mathbb{R}^{(D-d)\times k}$ is trained to produce a prediction, $U_\alpha \hat{\Theta}_\alpha \boldsymbol{h}_\alpha$.

Loss Function. The softmax layer takes the weighted sum between the linear subspace predictor $V\Theta_\alpha \boldsymbol{h}_\alpha$, and the null space predictor $U_\alpha \hat{\Theta}_\alpha \boldsymbol{h}_\alpha$, as the input. We define the matrix

$$W_\alpha = \lambda V \Theta_\alpha + (1-\lambda) U_\alpha \hat{\Theta}_\alpha.$$

and obtain the final weighted classifier, $W_\alpha^T \boldsymbol{h}_\alpha$, where we use $\lambda \in [0,1]$ to control the weighting between the shared subspace and the null space. The parameter λ controls the inherent tradeoff between the variance reduction (through the shared subspace), and model independences (through the null spaces). The extreme case when $\lambda = 1$ means that the predictors are completely relying on the common subspace features, whereas with $\lambda = 0$ the commonality across the ensemble members is removed and all networks become unrelated. The final training objective becomes

$$\text{Objective}(V, U_{1:M}, \Theta_{1:M}, \hat{\Theta}_{1:M}, \mathsf{Net}_{1:M}) = \frac{1}{nM} \sum_{\alpha=1}^{M} \sum_{i=1}^{n} \ell(y_i, \sigma(W_\alpha^T \boldsymbol{h}_\alpha^i)). \tag{6.1}$$

with a slight abuse of notation, where $\boldsymbol{h}_\alpha^i$ is to mean $\mathsf{Net}_\alpha(\boldsymbol{x}_i)$. The objective (6.1) is minimized with respect to all weights of the neural network, including matrices V and $U_1, \ldots, U_M$.

6.3.1 Soft Orthonormality Constraint

In practice, training deep neural networks while ensuring the orthonormality of $\{\Theta_\alpha, \hat{\Theta}_\alpha\}_{\alpha=1..M}$ is hard since it requires the costly computation of singular value decomposition (SVD), especially considering the dimensionality of $\boldsymbol{h}_\alpha$ can be of thousands in real networks and the entire training can take up to hundreds of thousands iterations. We use a soft penalty for constraining the orthonormality introduced in [112], and integrate the optimization of $\{\Theta_\alpha, \hat{\Theta}_\alpha\}_{\alpha=1..M}$ into the learning objective as a whole. Specifically, the soft orthonormality constraint for any projection matrix Θ is given by

$$\min \|\Theta\Theta^T - I\|_F^2.$$

Furthermore, for any individual network j, the orthonormality between the subspace and null space can be achieved similarly through the soft constraint of

$$\min \|\Theta_\alpha \hat{\Theta}_\alpha^T\|_F^2,$$

so that each component in $\hat{\Theta}_\alpha$ is orthogonal to Θ_α. The entire regularizer can then be formatted as a sum of orthonormality constraint on each individual network, with additional penalty on the model complexity.

Regularizer. The regularizer becomes

$$\begin{aligned} R(\Theta_{1:M}, \hat{\Theta}_{1:M}, U_{1:M}, V) = & \frac{\gamma}{M} \sum_{\alpha=1}^{M} \Big(\underbrace{\|\Theta_\alpha \Theta_\alpha^T - I\|_F^2}_{\text{Orthonormality}} + \underbrace{\|\Theta_\alpha \hat{\Theta}_\alpha^T\|_F^2 + \|\hat{\Theta}_\alpha \hat{\Theta}_\alpha^T - I\|_F^2}_{\text{Null space constraint}} \Big) \\ & + \frac{1}{M} \sum_{\alpha=1}^{M} \Omega(\Theta_\alpha, \hat{\Theta}_\alpha, U_\alpha, V), \end{aligned} \tag{6.2}$$

where $\Omega(\cdot)$ is a standard weight-decay regularization term on the weight parameters in order for controlling the model complexity; and γ the coefficient regulating the orthonormality.

The entire set of projection matrices can be optimized directly through any non-convex optimization algorithm (e.g., stochastic gradient descent) used in deep architectures. This advances previous work studying the structural learning under the setting of linear models [13, 12, 26] which uses an alternating structural optimization (ASO) procedure (i.e., performing the learning objective optimization and SVD computation in an alternating fashion).

Training Objective. Our final training objective can be formulated as the sum of Objective and regularizer

$$\arg \min_{V, U_{1:M}, \Theta_{1:M}, \hat{\Theta}_{1:M}, \mathsf{Net}_{1:M}} \text{Objective}(V, U_{1:M}, \Theta_{1:M}, \hat{\Theta}_{1:M}, \mathsf{Net}_{1:M}) + R(\Theta_{1:M}, \hat{\Theta}_{1:M}, U_{1:M}, V) \tag{6.3}$$

6.4 Experiments

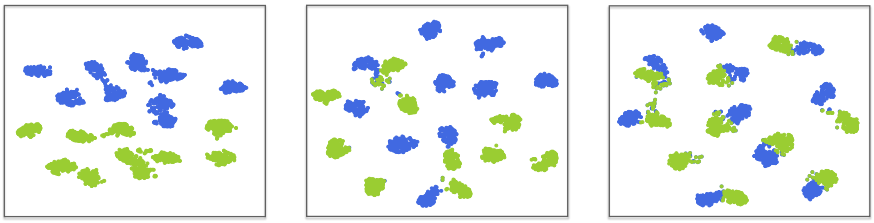
Our method is implemented in Tensorflow [53]. The Python code for reproducing all the following experiments is available at `https://github.com/yixuanli/sen`. We first demonstrate the technique for classification with fully connected networks (FCNs). We visualize our method 'under-the-hood', showing that each network truly learns an aligned common subspace, while learning a network-specific null space. We then demonstrate the effectiveness of our technique for classification. Finally, we show that our technique applies just as well to convolutional neural networks (CNNs).

6.4.1 Fully Connected Neural Networks

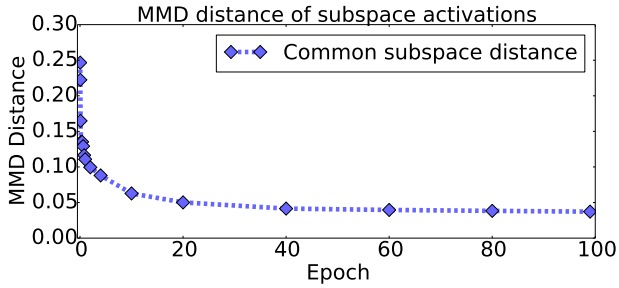
Network Configuration and Optimization. We begin by describing the configuration of our network and our optimization scheme. We train two fully connected networks Net_1 and Net_2 using our method. Each individual network consists of three hidden layers of $1,000$ rectified linear units, followed by a softmax output layer. The aligned subspaces $\boldsymbol{s}_1, \boldsymbol{s}_2$ have dimensionality $d = 64$ and are placed after the last hidden layer as described in the previous section and Figure 6.3. We train all models for 100 epochs (where an epoch is one pass over the training set) using mini-batch stochastic gradient descent with mini-batch size 100 and dropout probability 0.2. We use an initial learning rate 0.05 and drop after every 10 epochs with a decay rate 0.9. We fix the imitation parameter to $\lambda = 0.7$, and the orthonormality regularization coefficient to $\gamma = 200$.

Visualizing the Aligned Subspace. To show that the subspace representations $\boldsymbol{s}_1, \boldsymbol{s}_2$ are indeed aligned among networks Net_1 and Net_2 as the optimization proceeds, we visualize each subspace using t-Stochastic Neighbor Embedding (t-SNE) [177]. Figure 6.4 (a)-(c) shows the t-SNE embeddings the representations $\boldsymbol{s}_1, \boldsymbol{s}_2$ computed on the `MNIST` dataset after different numbers of training epochs. Each point corresponds to one validation input, each color signifies a certain network, and each cluster corresponds to one of the 10 digit classes in `MNIST`.

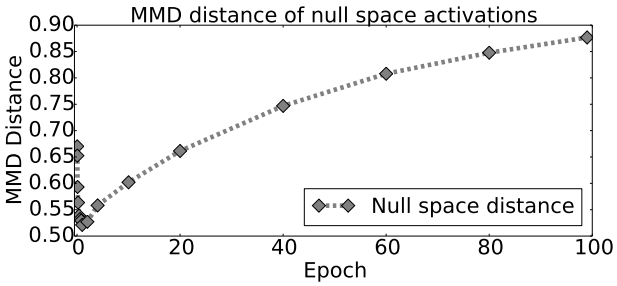
At the beginning of training, in each network, the subspace representations $\boldsymbol{s}_1, \boldsymbol{s}_2$ cluster by classes in different ways, seemingly learning distinct representations. After 10 epochs the representations of each network begin to overlap and after 100 epochs the two representations are well-aligned.



(a) Epoch 1 (b) Epoch 10 (c) Epoch 100



(d) MMD distance (subspace)



(e) MMD distance (null space)

Figure 6.4: **(a) - (c)** t-SNE [177] visualization of the subspace representations $\boldsymbol{s}_1, \boldsymbol{s}_2$ of two neural networks on the `MNIST` validation dataset. **(c) & (d)**: The MMD measured between the aligned representations $\boldsymbol{s}_1, \boldsymbol{s}_2$ and the null representations $\boldsymbol{s}_1^{\perp}, \boldsymbol{s}_2^{\perp}$, learned in Net_1 and Net_2.

To quantify how well the subspace representations align during training, we make use of a kernelized test-statistic that describes how much two distributions overlap, called the Maximum Mean Discrepancy (MMD) [66]. The MMD distance is 0 when two distributions are identical to each other. We compute an empirical estimate of the MMD between the aligned and null subspace representations of each network, over the entire training set $\{\boldsymbol{s}_{1,i}, \boldsymbol{s}_{1,i}^{\perp}\}_{i=1}^{n}$ and $\{\boldsymbol{s}_{2,i}, \boldsymbol{s}_{2,i}^{\perp}\}_{i=1}^{n}$, for different training epochs. Figure 6.4 (c) shows the MMD between the aligned subspace representations $\boldsymbol{s}_1, \boldsymbol{s}_2$. We note that the MMD approaches 0 as training continues, indicating the the two representations become nearly identical. Figure 6.4 (d) shows the MMD between the null subspaces $\boldsymbol{s}_1^{\perp}, \boldsymbol{s}_2^{\perp}$. As we suspect, the MMD continually increases throughout training, indicating that each network learns its own representation.

Ensemble Training Performance. Figure 6.5 shows a comparison of our technique Subspace Ensemble Networks(SEN) against traditional ensembling of neural networks. We evaluate our fully approach on five datasets which are modifications of the classic digit recognition dataset `MNIST`: `ROT`, the `MNIST` dataset in which digits are rotated arbitrarily, `BG-IMG` & `BG-RAND` superimpose various backgrounds on the digit images, and `BG-IMG-ROT` which performs includes background addition and digit rotation. Each variation set has $12,000$ training and $50,000$ testing examples. For each dataset we show the performance of the standard ensemble and SEN over different ensemble sizes. On `MNIST` we note that the standard ensemble (in green) reaches the lowest error 1.46% when using as many as 20 networks, whereas our approach (in blue) can yield test error of 1.44% with *only* 2-network ensemble. When training ensemble of *only* 2 networks with shared subspace, our approach can consistently outper-

form the standard ensemble approach with 20 networks.

Figure 6.6 (a) shows example of the validation curves during the first 4 training epochs of individually-trained Net_1 (in gray), and Net_1 with SEN (in blue, jointly trained with Net_2). We initialize both networks identically prior to training. When Net_1 is trained with a shared subspace, we observe a significantly faster convergence speed and also a higher final test accuracy.

6.4.2 Convolutional Neural Networks

We further demonstrate the compatibility of our method with modern convolutional neural networks. Many state-of-the-art architectures such as VGG [161], AlexNet [101] often consist of alternating convolutions and nonlinearities. These layers are followed by fully connected (FC) layers leading into a softmax classifier. Our subspace ensemble approach can be used directly after the high-dimensional FC layers in almost any deep architecture.

Network Configuration. We adopt the generic VGG architecture [161] with 3×3 kernels and train on the full CIFAR-10 dataset [98] with data augmentation. The images are cropped to size 24×24, with horizontal flipping and scaling. The model architecture and parameter setting can be found in the supplementary material (see Table 1). We train the model using standard gradient descent with a learning rate of 0.001 which exponentially decays every 50 epochs, we also fix a momentum to 0.9. We perform dropout after each pooling and fully connected layer, with a rate of 0.2. As is now common-practice batch normalization [78] is also used before each non-linearity. The standalone model achieves

an average test error of 7.97 ± 0.17% after 300 epochs, which matches roughly the performance reported in previous literature using the same architecture[3].

Optimization. To evaluate the effectiveness of our approach, we train two VGG networks with sharing subspace of dimensionality $d = 256$ on the last fully connected layer. We set the orthonormality regularization coefficient $\gamma = 100$ and $\lambda = 0.2$. As shown in Figure 6.5 (f) our method yields an average test error of 7.10 ± 0.16% with ensemble of 2 networks, is statistically identical to the performance of standard ensemble of 5 networks (6.98 ± 0.15%), while slightly worse than the 20 ensemble network[4].

Training Time. When training two networks using our method on a single `GeForce GTX TITAN X` with a batch size of 128, it takes anywhere from 0.3 to 0.6 seconds to process each mini-batch. The computation overhead caused by the subspace sharing is almost negligible compared to the entire forward and backward process. The training of both networks can be further parallelized for the remaining layers, which can reduce the computation time to nearly the same as individual deep networks.

Finally, we show the relative test error reduction averaged across all the six datasets in Figure 6.6 (b). With ensemble of 2 networks, our SEN approach can reduce the test error by 13.55% on average, whereas standard ensemble reduces by only 7.69% on average with 20 networks.

[3] http://torch.ch/blog/2015/07/30/cifar.html

[4] We suspect this is due to the parameters of subspace dimensionality d and weight coefficient λ are chosen suboptimally.

6.4.3 Sensitivity Analysis on λ and d

As λ controls the importance of shared predictive structure among classifiers, we further evaluate how the performance changes with respect to different λ. We train SEN with two networks on `MNIST` with increasing λ from 0 to 1. Figure 6.7 (a) displays the test error and variances with respect to λ. A salient observation is that our method can consistently outperform the standard ensemble with the same number of networks, invariant to the choice of λ. When comparing against standard ensemble of 20 networks, λ between 0.8 and 0.9 can yield good performance with low variance and error. We perform similar experiment for investigating the effect the subspace dimensionality d (see Figure 6.7 (b)). We fix λ to be 0.7 and choose a set of exponentially increasing number of $d = \{4, 8, 16, 32, 64, 128, 256\}$. We find the performance degrades when d becomes too large.

6.4.4 Bias-Variance Analysis of Ensemble Member Performance

We empirically validate the bias-variance hypothesis illustrated in Figure 6.2. To show that SEN can effectively bias the ensemble members toward a common classifier with reduced variance, we report in Table 6.1 (left column) for each dataset the average test error of 10 independently trained networks. We then randomly pair these networks into ensemble of 2 networks and train each of the 5 pairs with subspace sharing, and compute the test error averaged across all the 10 networks individually (see Table 6.1, right column). When trained with subspace sharing, the average performance of each individual networks is

largely improved compared to that of independent training.

Dataset	**Standard**	**SEN**
MNIST	1.58 ± 0.04%	1.50 ± 0.04%
BG-RAND	16.47 ± 0.40%	14.62 ± 0.24%
BG-IMG	19.19 ± 0.16%	18.24 ± 0.11%
ROT	12.54 ± 0.11%	11.69 ± 0.10%
BG-IMG-ROT	48.34 ± 0.23%	46.62 ± 0.21%
CIFAR-10	7.97 ± 0.17%	7.72 ± 0.15%

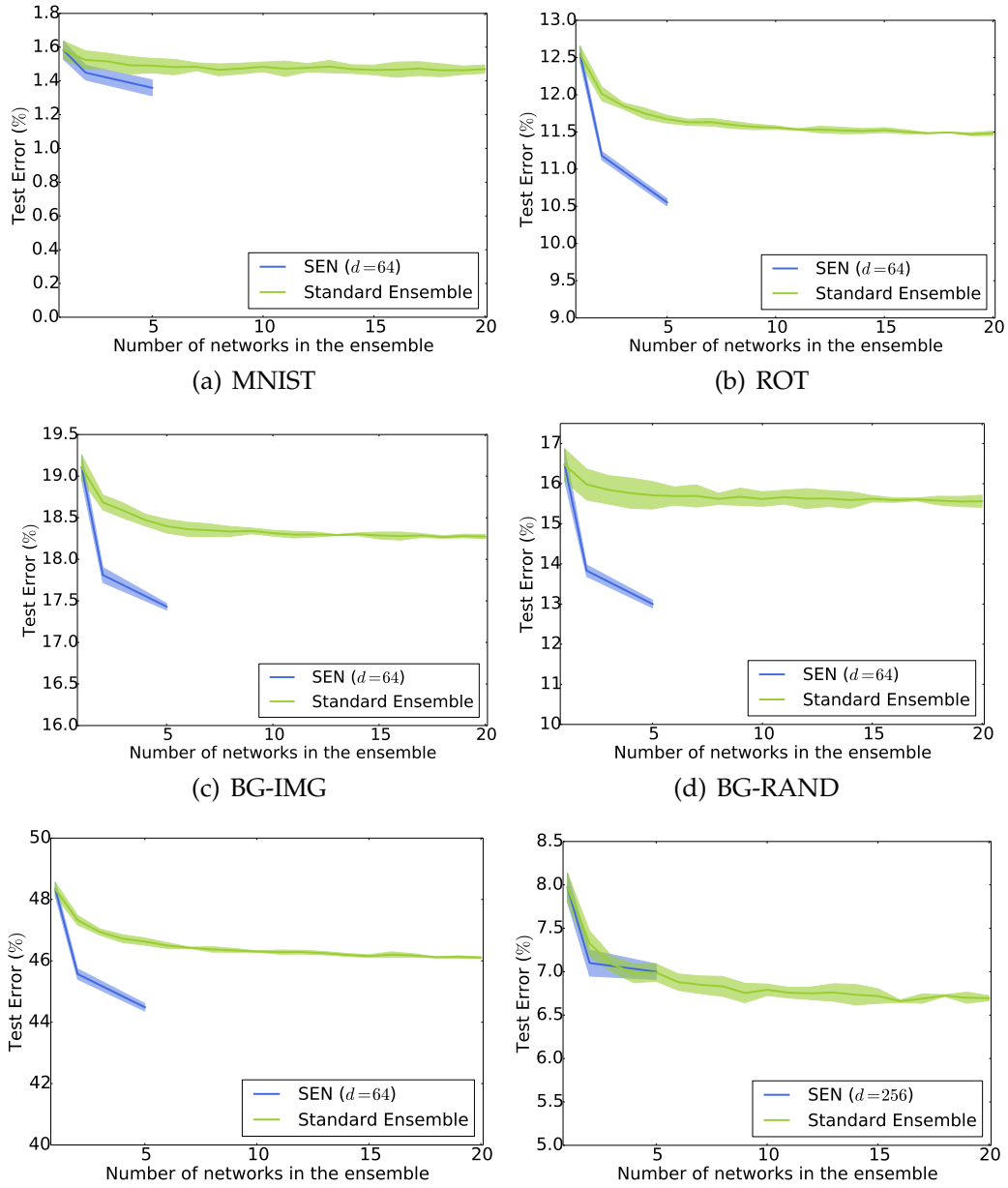
Table 6.1: Individual network test error (%) on six benchmark datasets.

6.5 Discussion and Future Work

We have introduced a novel learning paradigm, *Subspace Ensemble Networks* (SEN), for improving the generalization performance of small deep network ensembles. SEN provides a framework to trade-off the variance of individual deep networks through decomposition the internal representation into two components: (1) a shared representation subspace, that can be aligned across an ensemble, (2) a null space orthogonal to the shared subspace, that can learn network-specific features and make unique contribution to the ensemble compilation. SEN achieves better ensemble performance with drastic reductions in number of models. In 5 out of 6 data sets SEN with only 2 classifiers significantly outperforms conventional ensembles of 20 deep nets. When trained with subspace sharing, the average performance of each individual networks is also largely improved. SEN is compatible with almost any state-of-the-art neural architectures, and can be easily used as a plug-in to existing networks.

Our findings open up new future research directions, for example, (1) Domain-adaptation: our model can be extended to train multiple networks on

different datasets in the related domain. Similar approach can be used for distilling the common feature subspace between the source domain and target domain. (2) Multi-task learning: our method allows training different tasks while discovering the common predictive structure among them.

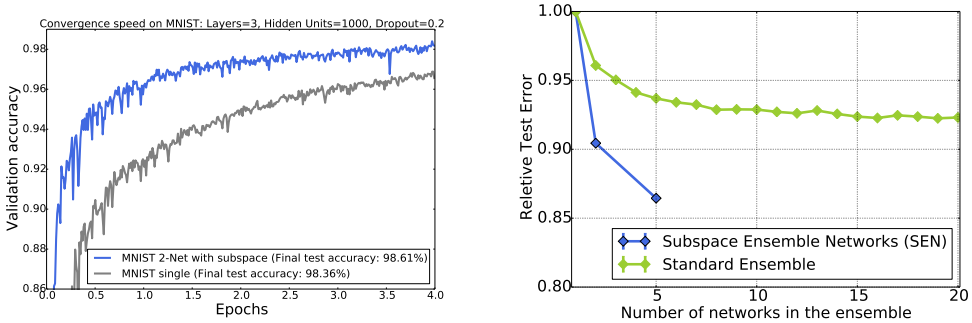


(a) MNIST (b) ROT

(c) BG-IMG (d) BG-RAND

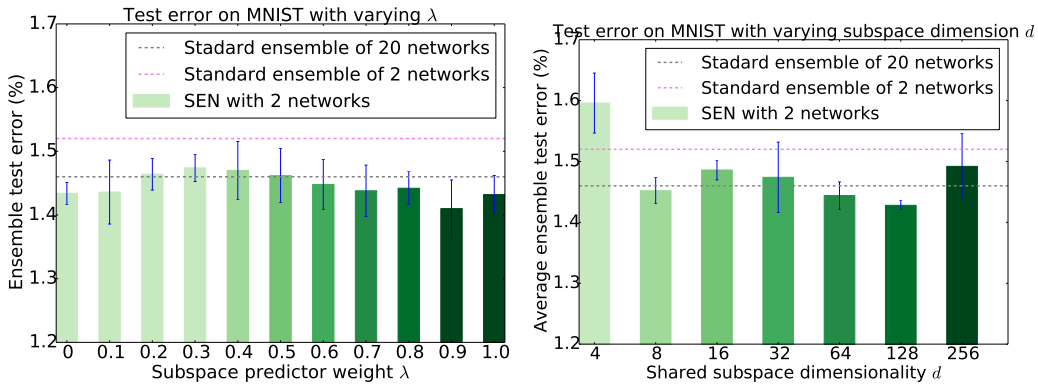
(e) BG-IMG-ROT (f) CIFAR-10

Figure 6.5: Comparison of ensemble test error (%) between our method (**blue**) and standard ensemble (**green**) on different benchmark datasets. We vary the number of networks from 1 to 20 in standard ensemble, and compare against our approach with ensemble of either 2 or 5 networks. The variances of the test errors are indicated by the shaded regions.



(a) Convergence speed on MNIST (b) Relative error reduction

Figure 6.6: **(a)** Comparison of convergence speed at the beginning of training on MNIST dataset. **Gray**: the validation curve when trained on MNIST standalone. **Blue**: the validation curve when trained on two neural networks with SEN. The subspace has dimensionality of 64. The curves shown here are both with the same weight initialization. **(b)** Comparison of the relative test error reduction averaged across all the six datasets.



(a) Sensitivity w.r.t. λ (b) Sensitivity w.r.t d

Figure 6.7: Sensitivity analysis on the subspace weight parameter λ and subspace dimensionality d respectively. The average test error and variance is shown for each parameter setting.

CHAPTER 7

EFFICIENT TRAINING OF DEEP NEURAL NETWORK ENSEMBLES

This section is written in collaboration with Gao Huang, Geoff Pleiss, Zhuang Liu, John Hopcroft and Kilian Weinberger. The work has been published in the 5th International Conference on Learning Representation in 2017.

7.1 Introduction

Stochastic Gradient Descent (SGD) [27] and its accelerated variants [92, 50] have become the de-facto approaches for optimizing deep neural networks. The popularity of SGD is typically attributed to its ability to avoid and even escape spurious saddle-points and local minima [44]. Although this ability is generally considered positive, in this paper we argue that there may in fact be some value to these local minima and that simply leaving them on the way-side may be outright wasteful.

Although deep networks typically never converge to a global minimum, there is a notion of "good" and "bad" local minima in terms of generalization. [91] argue that a local minimum generalizes well if it has a flat basin. SGD can avoid sharper spurious local minima because its gradients are computed from small mini-batches and are therefore inexact [91]. If the learning-rate is sufficiently large, there will be enough intrinsic random motion across gradient steps that the optimizer will not be trapped in any of the sharper local minima along its optimization path. If however the learning rate is sufficiently small, then the model will converge into the closest local minimum. These two very

different behaviors of SGD are typically exploited in different phases of the optimization [69]. Initially the learning rate is kept high to move into the general vicinity of a flat local minimum. Once this search has reached a stage in which the optimization makes no more progress, the learning-rate is dropped (once or twice), which triggers a descent and ultimately a convergence into the final local minimum.

It is well established [90] that the number of possible local minima grows exponentially with the number of parameters—of which modern neural networks can have millions. It is therefore not surprising that two identical architectures optimized from different initializations or even just with a different order of minibatches will converge to different solutions. Although different local minima often lead to very similar error rates, the corresponding neural networks tend to make very different mistakes. This diversity can be exploited through ensembling, in which multiple neural networks are trained from different initializations and then combined with majority voting or averaging [34]. Ensembling often leads to drastic reductions in error rates. In fact, most high profile competitions, e.g. Imagenet [47] or Kaggle[1], are won by ensembles of deep learning architectures. Despite its obvious advantages, the use of ensembling for deep networks is not nearly as wide-spread as for other algorithms. One likely reason for this lack of adaptation may be the additional training cost incurred by ensembling. Training deep networks is computationally expensive and can last for weeks, even on high performance hardware with GPU acceleration. As the training cost for ensembles increases linearly, ensembles can quickly becomes uneconomical for most researchers without access to industrial scale computational resources.

[1] www.kaggle.com

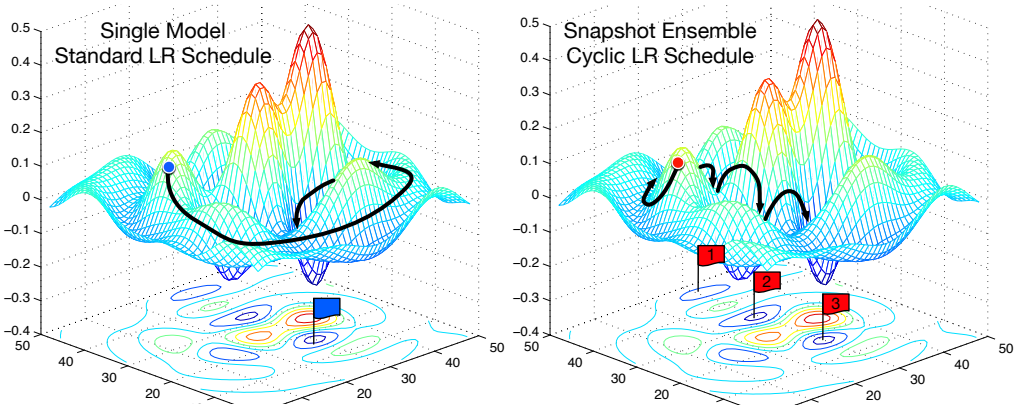


Figure 7.1: **Left:** Illustration of SGD optimization with a typical learning rate schedule. The model converges to a minimum at the end of training. **Right:** Illustration of Snapshot Ensembling optimization. The model undergoes several learning rate annealing cycles, converging to and escaping from multiple local minima. We take a snapshot at each minimum for test time ensembling.

In this paper we propose a method to achieve the seemingly contradictory goal to obtain an ensemble of multiple neural networks *without incurring any additional training costs*. Our approach leverages the non-convex nature of neural networks and the ability of SGD to converge into and escape from local minima on demand. Most importantly, it is compellingly simple and straight-forward to implement. Instead of training M neural networks independently from scratch, we let SGD converge M times into local minima along its optimization path. Each time the model converges we save the weights and add the corresponding network to our ensemble. We then restart the optimization with a large learning rate to escape the current local minimum and continue the optimization. To achieve this cyclic learning, we adopt the aggressive cosine annealing procedure suggested by [125], which lowers the learning rate continuously following a cosine function. Because our final ensemble consists of snapshots of the optimization path, we refer to our approach as *Snapshot Ensembling*. Figure 7.1 presents a high-level overview of this method.

In contrast to traditional ensembles, the training time for the entire ensemble is identical to the time required to train a *single* traditional model. During testing time, one can evaluate and average the last (and therefore most accurate) m out of M models. Our approach is naturally compatible with other methods to improve the accuracy, such as data augmentation, stochastic depth [77], or batch normalization [79]. In fact, Snapshot Ensemble can even be ensembled, if for example parallel resources are available during training. In this case, an ensemble of K Snapshot Ensembles yields $K \times M$ models at K times the training cost.

We evaluate the efficacy of Snapshot Ensemble on three popular state-of-the-art deep learning architectures for object recognition: ResNet [1], Wide-ResNet [196], and DenseNet [76]. We show across four different data sets that Snapshot Ensemble almost always leads to substantial reductions in error rates at the same training costs. For example, on Cifar-10 and Cifar-100 it obtains error rates of 3.44% and 17.41% respectively.

7.2 Related Work

Neural networks ensembles have been widely studied and applied in machine learning [67, 103]. However, most existing works focuses on improving the generalization performance, while few of them address the cost of training ensembles.

As an alternative to traditional ensembles, so-called "implicit" ensembles have high efficiency during both training and testing [167, 178, 77, 162, 104]. The Dropout [167] technique creates an ensemble out of a single model by "drop-

ping" — or zeroing — random sets of hidden nodes during each mini-batch. At test time, no nodes are dropped, and each node is scaled by the probability of surviving during training. [167] claim that Dropout reduces overfitting by preventing the co-adaptation of nodes. An alternative explanation is that this mechanism creates an exponential number of networks with shared weights during training, which are then implicitly ensembled at test time. DropConnect [178] uses a similar trick to create ensembles at test time by dropping connections (weights) during training instead of nodes. The recently proposed Stochastic Depth technique [77] randomly drops layers during training to create an implicit ensemble of networks with varying depth at test time. Stochastic Depth has a strong regularization effect and tends to outperform Dropout on very deep networks [77]. Finally, Swapout [162] is a stochastic training method that generalizes Dropout and Stochastic Depth. From the perspective of model ensembling, it provides diversified network structures for model averaging. Our proposed method similarly trains only a single model; however, the resulting ensemble is explicit in that the models do not share weights. It is also be used in conjunction with implicit ensembling techniques.

Several recent publications focus on reducing the *test time cost* of ensembles, by transferring the "knowledge" of cumbersome ensembles into a single model [30, 72]. [72] propose to use an ensemble of multiple networks as the target of the single (smaller) network. In order to produce more informative guidance, a high temperature is used to soften the probability distribution over classes. Our proposed method is complementary to these works as we aim to reduce the *training* cost of ensembles rather than the test time cost.

There are several recently proposed training mechanisms to improve the

power of explicit ensembles. [107] propose a temporal ensembling method for semi-supervised learning, which achieves consensus among models trained with different regularization and augmentation conditions. Recently, [131] show that boosting can be applied to convolutional neural networks to create strong ensembles.

Perhaps most similar to our work is that of [173] and [188], who have explored creating ensembles from slices of the learning trajectory. [188] introduce the *horizontal and vertical* ensembling method, which combines the output of networks within a range of training epochs. More recently, [80] and [158] show the improvement by building an ensemble of the intermediate stages of model training. Our work differs from these prior works in that we ensure our snapshots are taken after a model has reached a local minimum, rather than taking snapshots before the model has converged. We believe this key insight allows us to leverage more power from our ensembles.

Our work is inspired by recent work of [125] and [163], who show that cyclical learning rates can be effective for training convolutional neural networks. The authors show that each cycle produces models which are (almost) competitive to those learned with traditional learning rate schedules in a fraction of training iterations. Although the model performance temporarily suffers when the learning rate cycle is restarted, the performance eventually surpasses that of the previous cycle after annealing the learning rate. The authors suggest that cycling perturbs the parameters of a converged model, which allows the model to find a better local minimum. Our work builds upon these recent findings in non-linear function optimization, but we are not concerned with speeding up or improving the training of a single model but instead try to extract an ensemble

of classifiers while following the optimization path of the final model.

7.3 Snapshot Ensembling

Snapshot Ensembling produces an ensemble of accurate and diverse models from a single training process. At the heart of Snapshot Ensembling is an optimization process which visits several local minima before converging to a final solution. We take model snapshots at these various minima, and average softmax predictions from each of them at test time.

Ensembles work best if their individual members have low test error and have little overlap in the set of examples they still misclassify. Almost all weight assignments along the optimization path of a neural network do not lead to low error models. In fact, it is commonly observed that the validation error drops significantly only after the learning rate has been reduced at least once, which is typically done after several hundred epochs. Our approach is inspired by the observation that training neural networks for fewer epochs—dropping the learning rate earlier—often increases the final error by only a small amount [125]. This seems to suggest, that the local minima along the optimization path are already starting to become promising in terms of generalization error after only several epochs.

Cyclical Cosine Annealing. In our search for viable local minima we follow a cyclic annealing schedule as proposed by **(author?)** [125]. We lower the learning rate at a very fast pace, encouraging the model to converge towards its first local minimum after as few as 50 epochs. The optimization is then continued at a larger learning rate, perturbing the model to snap out of its local minimum and

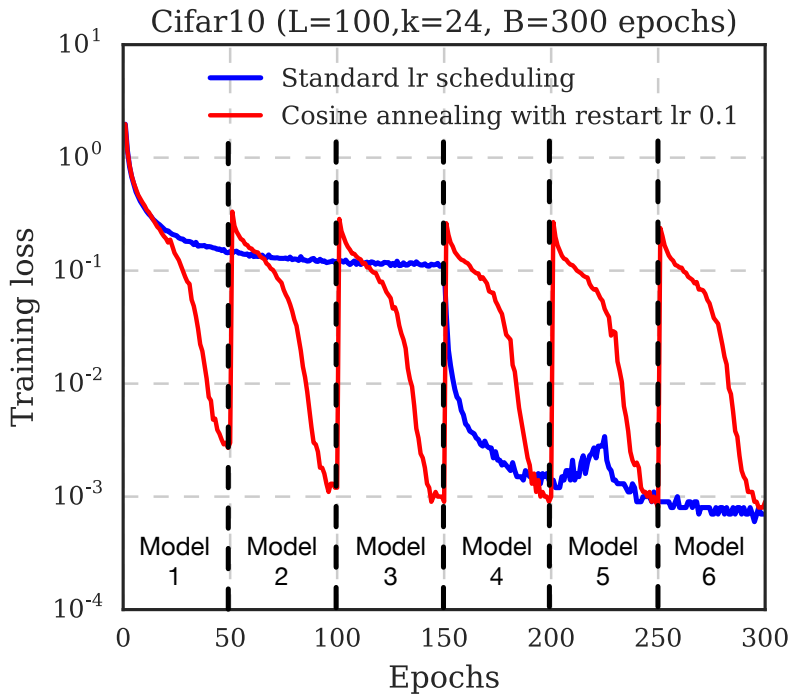


Figure 7.2: Training loss of 100-layer DenseNet on CIFAR10 when using standard learning rate (blue) and $M = 6$ cosine annealing cycles (red). The intermediate models, denoted by the dotted lines, are ensembled together at the end of training.

the annealing cycle starts over. Formally, the learning rate α has the form:

$$\alpha(t) = f\left(\mathrm{mod}\left(t-1, \lceil T/M \rceil\right)\right), \tag{7.1}$$

where t is the iteration number, T is the total number of training iterations, and f is a monotonically decreasing function. In other words, we split the training process into M cycles, each of which starts with a large learning rate value. Each cycle then undergoes a full annealing process before raising the learning rate at the beginning of the next cycle. The large learning rate $\alpha = f(0)$ in each stage gives the model enough energy to escape from a critical point, and the small learning rate $\alpha = f(T/M)$ will drive the model to a well behaved local minimum. The learning rate function in (7.1) can be any monotonically decreasing function.

We adopt the shifted cosine function proposed by [125]:

$$\alpha(t) = \frac{\alpha_0}{2}\left(\cos\left(\frac{\pi \text{mod}(t-1, \lceil T/M \rceil)}{\lceil T/M \rceil}\right) + 1\right), \tag{7.2}$$

where α_0 is the initial learning rate. Intuitively, this function anneals the learning rate from its initial value $\alpha_0$to $f(T/M) = 0$ over the course of a cycle. Following [125], we update the learning rate at each iteration rather than at every epoch. This improves the convergence of short cycles, even when a large initial learning rate is used .

Snapshot Ensembling. Figure 7.2 depicts the training process using cyclic and traditional learning rate schedules. At the end of each training cycle, it is apparent that the model reaches a local minimum with respect to training the loss. Thus, before raising the learning rate, we take a "snapshot" of the model weights (indicated as vertical dashed black lines). After training M cycles, we have M model snapshots, $f_1 \ldots f_M$, each of which will be used in the final ensemble. It is important to highlight that the total training time of the M snapshots is the same as training a model with a standard schedule (indicated in blue). It is also fair to point out that the training loss under the standard schedule is eventually lower—however, as we will show in the next section, ensembling the M models more than compensates for that.

Ensembling at test-time. At test time, we compute the ensemble prediction by averaging the softmax outputs of the last m models. Let $\mathbf{x}$ be a test sample and let $f_i(\mathbf{x})$ be the softmax score of snapshot i. The output of the ensemble is a simple average of the last m models: $f_{Ensemble} = \frac{1}{m}\sum_0^{m-1} f_{M-i}(\mathbf{x})$. We always ensemble the last m models because the generalization error of the models tends to monotonically decrease.

	Method	C10	C100	SVHN	Tiny ImageNet
ResNet-110	Single model	5.52	28.02	1.96	46.50
	NoCycle Snapshot Ensemble	5.49	26.97	1.78	43.69
	SingleCycle Ensembles	6.66	24.54	1.74	42.60
	Snapshot Ensemble ($\alpha_0 = 0.1$)	5.73	25.55	**1.63**	40.54
	Snapshot Ensemble ($\alpha_0 = 0.2$)	**5.32**	**24.19**	1.66	**39.40**
Wide-ResNet-32	Single model	5.43	23.55	1.90	39.63
	Dropout	4.68	22.82	1.81	36.58
	NoCycle Snapshot Ensemble	5.18	22.81	1.81	38.64
	SingleCycle Ensembles	5.95	21.38	1.65	35.53
	Snapshot Ensemble ($\alpha_0 = 0.1$)	**4.41**	**21.26**	1.64	35.45
	Snapshot Ensemble ($\alpha_0 = 0.2$)	4.73	21.56	**1.51**	**32.90**
DenseNet-40	Single model	5.24*	24.42*	1.77	39.09
	Dropout	6.08	25.79	1.79*	39.68
	NoCycle Snapshot Ensemble	5.20	24.63	1.80	38.51
	SingleCycle Ensembles	5.43	22.51	1.87	38.00
	Snapshot Ensemble ($\alpha_0 = 0.1$)	4.99	23.34	**1.64**	37.25
	Snapshot Ensemble ($\alpha_0 = 0.2$)	**4.84**	**21.93**	1.73	**36.61**
DenseNet-100	Single model	3.74*	19.25*	-	-
	Dropout	3.65	18.77	-	-
	NoCycle Snapshot Ensemble	3.80	19.30	-	-
	SingleCycle Ensembles	4.52	18.38		
	Snapshot Ensemble ($\alpha_0 = 0.1$)	3.57	18.12	-	-
	Snapshot Ensemble ($\alpha_0 = 0.2$)	**3.44**	**17.41**	-	-

Table 7.1: Error rates (%) on CIFAR-10 and CIFAR-100 datasets. All methods in the same group use the same amount of training time and cost. Results of our method are colored in blue, and best result for each network and each dataset is shown in **bold**. * indicates numbers directly taken from [76].

7.4 Experiments

We demonstrate the effectiveness of Snapshot Ensembles on several benchmark datasets and compare it with competitive baselines. We run all experiments with Torch 7 [41][2].

[2]Code to reproduce results is available at `https://github.com/gaohuang/SnapshotEnsemble`

7.4.1 Datasets

CIFAR. The two CIFAR datasets [99] consist of colored natural scene images sized at 32×32 pixels. CIFAR-10 (C10) and CIFAR-100 (C100)images are drawn from 10 and 100 classes respectively. For each dataset, there are 50,000 training images and 10,000 images reserved for testing. A standard data augmentation scheme is used as in [124, 153, 114, 166, 168, 69, 77, 110]: the images are first zero-padded with 4 pixels on each side, then randomly cropped to again produce 32×32 images; half of the images are then horizontally mirrored.

SVHN. The Street View House Numbers (SVHN) dataset [135] contains 32×32 colored digit images from Google Street View, with 10 different classes for each digit. There are 73,257 images in the training set and 26,032 images in the test set. Following common practice [159, 65, 76], we withhold 6,000 training images for validation, and train on the remaining images without data augmentation.

Tiny ImageNet. The Tiny ImageNet dataset[3] consists of a subset of ImageNet ILSVRC images [47], with 200 classes. Each class has 500 training images, 50 validation images and 50 test images. Each image is resized to 64×64, with standard data augmentation including random crops, horizontal mirroring, and altering RGB intensities [102].

ImageNet. The ILSVRC 2012 classification dataset [47] consists 1.2 million training images from 1000 different classes, and there are 50,000 images for validation. We adopt the same data augmentation scheme for the training images as in [69, 76], and apply a 224×224 center crop to images at test time.

[3] `https://tiny-imagenet.herokuapp.com`

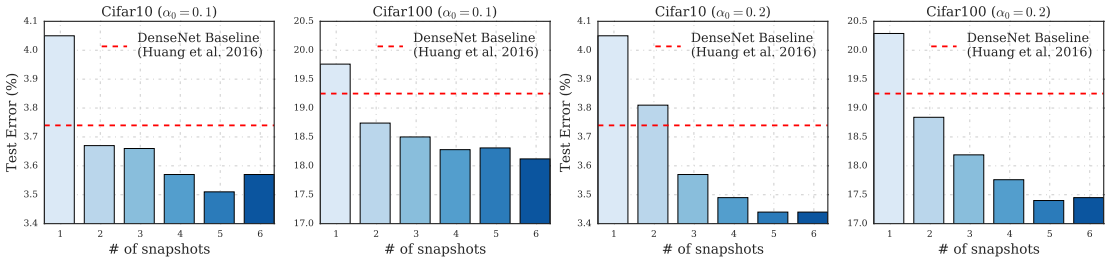


Figure 7.3: Snapshot Ensemble performance on CIFAR-10 and CIFAR-100 when trained on DenseNet-100 with restart learning rate 0.1 (left two) and 0.2 (right two) with $M = 6$ annealing cycles (50 epochs per each).

7.4.2 Training Setting

Architectures. We benchmark on several state-of-the-art architectures including residual networks (*ResNet*) [1], *Wide ResNet* [196] and *DenseNet* [76]. For ResNet, we use the original 110-layer network introduced by [69]. The Wide-ResNet) is a 32-layer ResNet with each of its convolutional layer four times wider (in terms of the number of output channels) than standard ResNets. For DenseNet, our large model in use is consistent with the original setup in [76], with depth $L = 100$, growth rate $k = 24$. In addition, we also evaluate our method on a small DenseNet, with depth $L = 40$ and $k = 12$. For Tiny ImageNet, we add a stride of 2 to the first convolution of each model to downsample the images to a 32×32 size. For ImageNet, the 50-layer ResNet proposed in [69] is used.

We use a batch size 64 for all the architectures and on all datasets, except the ResNet-110 and Wide-ResNet are trained with batch size 128 on Tiny ImageNet, and the ResNet-50 for ImageNet is trained with batch size 256.

Baselines. Snapshot Ensembles incur the training cost of a single model; therefore, we compare with baselines that require the same amount of train-

ing. Firstly, we compare against a **Single Model** trained with a standard learning rate schedule, dropping the learning rate from 0.1 to 0.01 halfway through training, and then to 0.001 when training is at 75%. Additionally, to compare against implicit ensembling methods, we test against a single model trained with **Dropout**. This baseline uses the same learning rate as above, and drops nodes during training with a probability of 0.2.

We then test variants of the **Snapshot Ensemble** algorithm trained with the cyclic cosine cycle as described by Equation (7.2). We test models with the max learning rate α_0 set to 0.1 and 0.2. In both cases, we divide the training process into six contingent learning rate cycles. We take a model snapshot at the end of each training cycle. Additionally, we train a Snapshot Ensemble with a non-cyclic learning rate schedule. This **NoCycle Snapshot Ensemble**, which uses the same schedule as the Single Model and Dropout baselines, is meant to highlight the impact of cyclic learning rates for our method. To accurately compare with the cyclical Snapshot Ensembles, we take six snapshots equally spaced throughout the training process. Finally, we compare against **SingleCycle Ensembles**, a Snapshot Ensemble variant in which the network is re-initialized at the beginning of every cosine learning rate cycle, rather than using the parameters from the previous optimization cycle. This baseline essentially creates a traditional ensemble, yet each network only has $1/M$ of the typical training time. This variant is meant to highlight the tradeoff between model diversity and model convergence. Though SingleCycle Ensembles should in theory explore more of the parameter space, the models do not benefit from optimization of previous cycles. For each of the variants, we test the ensemble by feeding samples through each model snapshot and averaging the softmax outputs.

Method	**Val. Error (%)**
Single model	24.01
Snapshot Ensemble ($M = 2$)	23.33
Snapshot Ensemble ($M = 3$)	23.96

Table 7.2: Top-1 error rates (%) on ImageNet validation set using ResNet-50 with varying number of cycles.

Training Cost. The DenseNet-40 and DenseNet-100 baseline models are trained for a total of $B = 300$ epochs (150 for Tiny ImageNet). Snapshot variants were trained with $M = 6$ cycles of $B/M = 50$ epochs each (25 for Tiny ImageNet). Following the original training budget in [69, 196], we train ResNet and Wide ResNet models for 200 epochs (150 for Tiny ImageNet). Snapshot variants of ResNet and Wide ResNet were trained with 5 cycles of 40 epochs each (6 cycles of 25 epochs for Tiny ImageNet). On SVHN dataset, we consistently train all the models using a total training budget of $B = 40$ epochs. Snapshot variants in this case are trained with 5 cycles of 8 epochs each. On ImageNet, the total training budget is $B = 90$ epochs, and we experiment with 2 and 3 training cycles.

7.4.3 Snapshot Ensemble results

Accuracy. The main results are summarized in Table 7.1. In all but one experiment, Snapshot ensembles achieve lower error than any of the baseline methods. Most notably, Snapshot Ensembling yields an error rate of 17.41% on CIFAR-100 using large DenseNets, far outperforming the state-of-the-art record of 19.25% under the same training cost and architecture [76]. Our method has the most success on CIFAR-100 and Tiny ImageNet, which is likely due to the complexity of these datasets. The softmax outputs for these datasets are high

dimensional due to the large number of classes, making it unlikely that any two models make the same predictions. Snapshot Ensembling is also capable of improving the competitive baselines for CIFAR-10 and SVHN, reducing error by 1% and 0.4% respectively with the Wide ResNet architecture.

The NoCycle Snapshot Ensemble generally has little effect on performance, and in some instances even *increases* the test error. This highlights the need for a cyclical learning rate to discover solutions for useful ensembling. The SingleCycle Ensemble has similarly mixed performance. This demonstrates that Snapshot Ensembles work best when utilizing information from previous cycles. Effectively, Snapshot Ensembles strike a balance between model diversity and optimization. In some cases, e.g., DenseNet-40 on CIFAR-100, the SingleCycle Ensemble is competitive compared to Snapshot Ensemble. However, as the model size increases to 100 layers, it becomes less attractive. This is because training larger models from scratch for a short learning rate cycle tends to give less accurate solutions.

Table 7.2 shows Snapshot Ensemble results on ImageNet. The Snapshot Ensemble with $M = 2$ achieves 23.33% validation error, outperforming the single model baseline with 24.01% validation error. It appears that 2 cycles is the optimal choice for the ImageNet dataset. We hypothesize that allocating fewer than $B/2 = 45$ epochs per training cycle is insufficient for the ResNet-50 model to converge to a good local minimum on such a large dataset.

Ensemble Size. In some applications, it may be beneficial to vary the size of the ensemble *dynamically* at test time depending on available resources. Figure 7.3 displays the performance of CIFAR DenseNets as the effective ensemble size, m, is varied. Each ensemble is comprised of snapshots from later cycles, as

M	Test Error (%)
2	22.92
4	22.07
6	21.93
8	21.89
10	22.16

Table 7.3: Error rates (%) on CIFAR-100 using DenseNet-40 with varying number of cycles.

these snapshots have received the most training and converge to better minima. Although ensembling more models generally gives better performance, we observe significant drops in error when the second and third models are added to the ensemble. In most cases, an ensemble of two models outperforms the baseline model.

Restart Learning Rate. The effect of restart learning rate can be observed in Figure 7.3. The left two plots show the performance when using restart learning rate of $\alpha_0 = 0.1$ at the beginning of each cycle, and the bottom panel shows that of $\alpha_0 = 0.2$ restart learning rate. In most cases, we find that larger restart learning rate can lead to better ensemble performance, presumably due to larger diversity introduced by the stronger perturbation inbetween cycles.

Varying Number of Cycles Given a fixed training budget, there is a trade-off between the number of learning rate cycles and their length. Therefore, we investigate how the number cycles M affects the ensemble performance, give a training budget of $B = 300$ epochs. We train a 40-layer DenseNet on CIFAR-100 dataset with an initial learning rate of $\alpha_0 = 0.2$. As shown in Table 7.3, our method is relatively robust with respect to different values of M. At the extremes, $M = 2$ and $M = 10$, we find a slight degradation in performance, as the cycles are either too few or too short. In practice, we find setting M to be

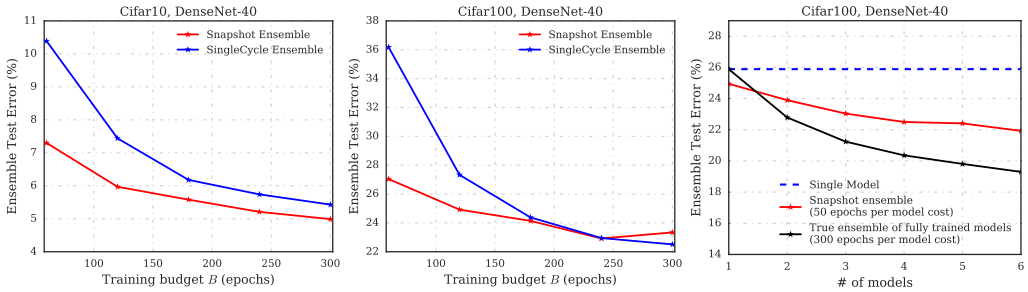


Figure 7.4: **Left:** Snapshot Ensembles under different training budget on CIFAR10. **Middle:** Snapshot Ensembles under different training budget on CIFAR100. **Right:** Performance comparison of Snapshot Ensembles with true ensembles.

$4 \sim 8$ works reasonably well.

Varying Training Budget. The left and middle panels of Figure 7.4 show the performance of Snapshot Ensemble (we fix $M = 6$) as a function of training budget. We train a 40-layer DenseNet on CIFAR-10 and CIFAR-100, with an initial learning rate of $\alpha_0 = 0.1$, and the total number of epochs ranging from 60 to 300 (with step size 60). We can observe a clear trend that Snapshot Ensemble becomes more accurate as the training budget increases. However, we highlight that with small training budget (corresponds to the left part of the curves), Snapshot Ensemble can still yield competitive results, while the performance of the SingleCycle Ensemble degrades dramatically. The results also suggest that Snapshot Ensemble is most useful when there is limited training budget.

Comparison with True Ensemble. We compare Snapshot Ensemble with the expensive traditional ensemble method. The right panel of Figure 7.4 shows the test rates of DenseNet-40 on CIFAR-100. The true ensemble method averages models that are trained with 300 full epochs, each with different weight initializations. Given the same number of models at test time, the error rate of the true ensemble can be seen as an lower bound of our method. Our method achieves

performance that compares favorably with ensembling of 3 independant models, but with the training cost of one model.

7.4.4 Diversity of model ensembles

Parameter Space. We hypothesize that the cyclical learning rate schedule creates snapshots which are not only accurate but also diverse with respect to model predictions. We qualitatively measure this diversity by visualizing the local minima they fall into. For that purpose we linearly interpolate snapshot models, as described by [64]. Let $J(\theta)$ be the test error of a model using parameters θ. Given θ_1 and θ_2 — the parameters from models 1 and 2 respectively — we can compute $J(\theta)$ for various combinations of these parameters: $\theta = \lambda(\theta_1) + (1 - \lambda)(\theta_2)$, where λ is a mixing coefficient. Setting λ to 1 results in a parameters that are entirely θ_1 while setting λ to 0 give the parameters θ_2. By sweeping the values of λ, we can examine an linear slice of the parameter space. Two models with similar convergence point will have smooth parameter interpolations, whereas the interpolation for models at different minima will likely be non-convex and show a "bump" around $\lambda = 0.5$.

In Figure 7.5, we draw interpolations between the final model of DenseNet-40 (sixth snapshot) and all intermediate snapshots. The left two of the plots denote Snapshot Ensembles trained with a cyclical learning rate, while the right two plots denote NoCycle Snapshot Models. $\lambda = 0$ represents a model which is entirely snapshot parameters, while $\lambda = 1$ represents a model which is entirely the final parameters. From this figure, it is clear that there are differences between cyclical and non-cyclical learning rate schedules. Firstly, all of the cycli-

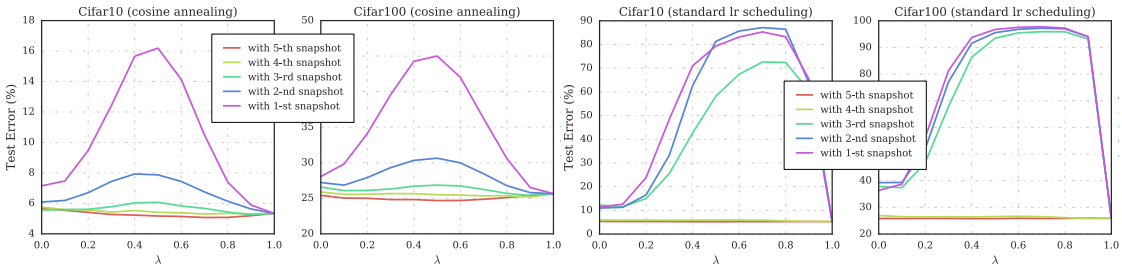


Figure 7.5: Interpolations in parameter space between the final model (sixth snapshot) and all intermediate snapshots. $\lambda = 0$ represents an intermediate snapshot model, while $\lambda = 1$ represents entirely final model parameters. **Left:** Snapshot Ensemble, with cosine annealing cycles ($\alpha_0 = 0.2$ every $B/M = 50$ epochs). **Right:** NoCycle Snapshot Ensemble, (two learning rate drops, snapshots every 50 epochs).

cal snapshots achieve roughly the same error as the final cyclical model, as the error is similar for $\lambda = 0$ and $\lambda = 1$. Additionally, it appears that most snapshots do not lie in the same minimum as the final model. Thus the snapshots are likely to misclassify different samples. Conversely, the first three snapshots achieve much higher error than the final model. This can be observed by the sharp minima around $\lambda = 1$, which suggests that mixing in any amount of the snapshot parameters will worsen performance. While the final two snapshots achieve low error, the figures suggests that they lie in the same minimum as the final model, and therefore likely add limited diversity to the ensemble.

Activation space. To further explore the diversity of models, we compute the pairwise correlation of softmax outputs for every pair of snapshots. Figure 7.6 displays the average correlation for both for cyclical snapshots and for non-cyclical snapshots. Firstly, there are large correlations between the last 3 snapshots of the non-cyclical training schedule (right). These snapshots are taken after dropping the learning rate, suggesting that each snapshot has converged to the same minimum. Though there is more diversity amongst the ear-

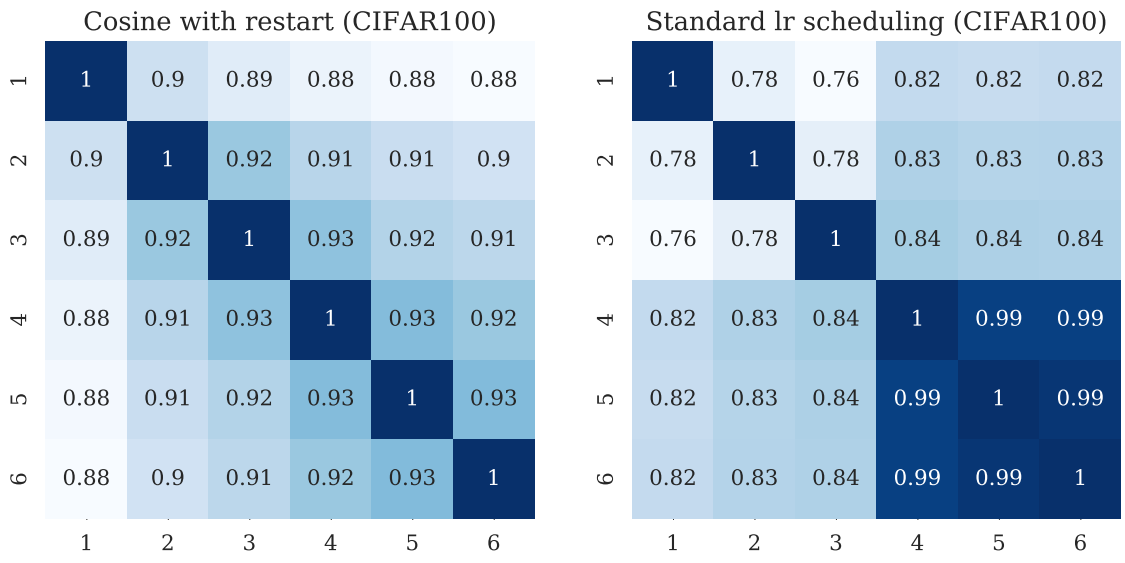


Figure 7.6: Pairwise correlation of softmax outputs between any two snapshots for DenseNet-100. **Left:** Snapshot Ensemble, with cosine annealing cycles (restart with 0.2 every 50 epochs). **Right:** NoCycle Snapshot Ensemble, (two learning rate drops, snapshots every 50 epochs).

lier snapshots, these snapshots have much higher error rates and are therefore not ideal for ensembling. Conversely, there is less correlation between all cyclical snapshots (left). Because all snapshots have similar accuracy (as can be seen in Figure 7.5), these difference in predictions can be exploited to create effective ensembles. As expected, correlations are strongest between consecutive snapshots.

7.5 Discussion

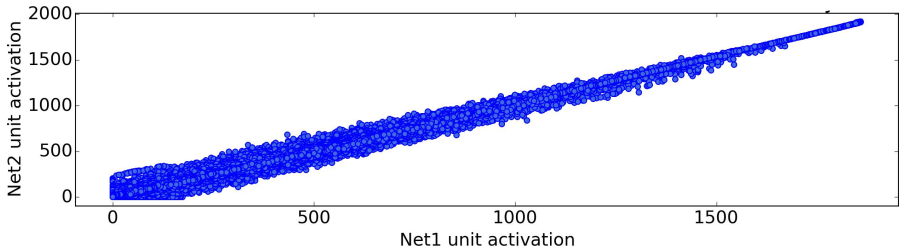
We introduce Snapshot Ensemble, a simple method to obtain ensembles of neural networks without any additional training cost. Our method exploits the ability of SGD to converge to and escape from local minima, which allows the

model to visit several accurate solutions over the course of training. We harness this power with cyclical learning rate schedule proposed by **(author?)** [125], saving snapshots of the models at each point of convergence. We show in several experiments that all acquired snapshots are accurate, yet produce different predictions from one another, and therefore are ideal for test-time ensembles. Ensembles of these snapshots significantly improve the state-of-the-art on the CIFAR-10, CIFAR-100 and SVHN datasets. Future work will explore combining Snapshot Ensembles with traditional ensembles. In particular, we will investigate how to balance growing an ensemble with new models (with random initializations) and refining existing models with further training cycles under a fixed training budget.

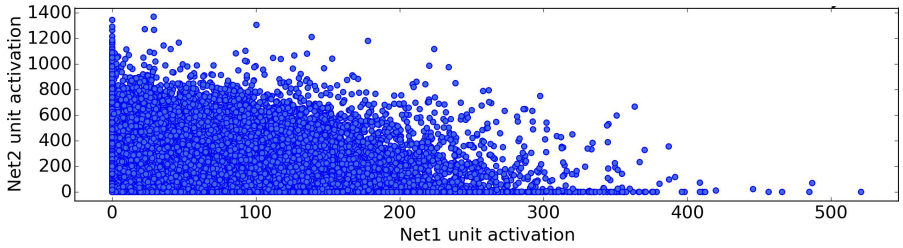
APPENDIX A

APPENDIX FOR CHAPTER 4

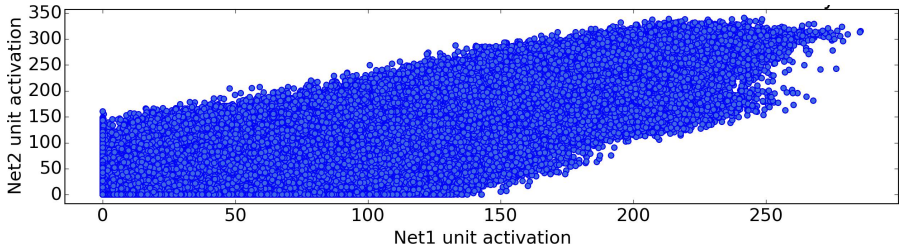
A.1 Activation Values: High Correlation vs. Low Correlation



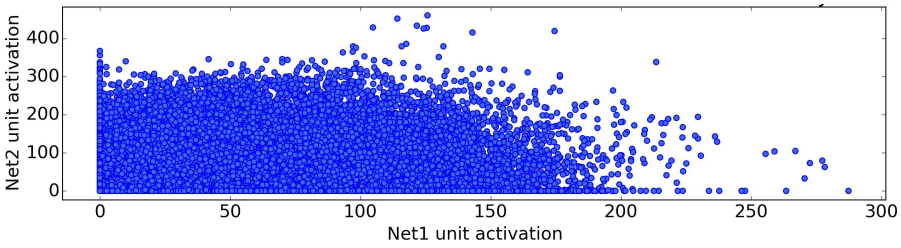
(a) conv1, high correlation $c^{(1,2)}_{1,34,48}$: 0.995



(b) conv1, low correlation $c^{(1,2)}_{1,2,74}$: 0.178



(c) conv2, high correlation $c^{(1,2)}_{2,122,236}$: 0.92



(d) conv2, low correlation $c^{(1,2)}_{2,30,26}$: 0.24

Figure A.1: Activation values are computed across 5,000 randomly sampled images and all spatial positions (55×55 and 27×27 for layers conv1 and conv2, respectively). The joint distributions appear simple enough to suggest that a correlation measureis sufficient to find matching units between networks.

A.2 Additional Matching Results

Figure A.3 shows additional results of comparison of assignments produced by semi-matching and matching methods in conv2 – conv5. For each unit, both the semi-matching and matching are found, then the units are sorted in order of decreasing semi-matching value and both correlation values are plotted.

A.3 Does Relaxing the One-to-One Constraint to Find Many-to-Many Groupings Reveal More Similarities Between what Different Networks Learn?

Since the preceding sections have shown that neurons may not necessarily correspond via a globally consistent one-to-one matching between networks, we now seek to find many-to-many matchings between networks using a spectral clustering approach.

A.3.1 Neuron Similarity Graphs

We define three types of similarity graphs based on the correlation matrices obtained above.

Single-net neuron similarity graphs. Given a fully trained DNN X and a specified layer l, we first construct the *single-net neuron similarity graph* $G_{X,l} = (V, E)$. Each vertex v_p in this graph represents a unit p in layer l. Two vertices are connected by an edge of weight c_{pq} if the correlation value c_{pq} in the self-correlation matrix $\text{corr}(X_l, X_l)$ between unit p and unit q is greater than a certain threshold τ.

Between-net neuron similarity graphs. Given a pair of fully trained DNNs X and Y, the *between-net neuron similarity graph* $G_{XY,l} = (V, E)$ can be constructed in a similar manner. Note that $G_{XY,l}$ is a bipartite graph and contains twice as many vertices as that in $G_{X,l}$ since it incorporates units from both networks. Two vertices are connected by an edge of weight c_{pq} if the correlation value c_{pq} in the

between-net correlation matrix $\text{corr}(X_l, Y_l)$ between unit p in X and unit q in Y is greater than a certain threshold τ.

Combined similarity graphs. The problem of matching neurons in different networks can now be reformulated as finding a partition in the combined neuron similarity graphs $G_{X+Y,l} = G_{X,l} + G_{Y,l} + G_{XY,l}$, such that the edges between different groups have very low weights and the edges within a group have relatively high weights.

A.3.2 Spectral Clustering and Metrics for Neuron Clusters

We define three types of similarity graphs based on the correlation matrices obtained above (see Section A.3.1 for definition). Define $W_l \in \mathbb{R}^{2\mathcal{S}_l \times 2\mathcal{S}_l}$ to be the combined correlation matrices between two DNNs, X and Y in layer l, where w_{jk} is the entry at jth row and kth column of that matrix. $\mathcal{S}_l$ is the number of channels (units) in layer l. W_l is given by

$$W_l = \begin{bmatrix} \text{corr}(X_l, X_l) & \text{corr}(X_l, Y_l) \\ \text{corr}(X_l, Y_l)^\top & \text{corr}(Y_l, Y_l) \end{bmatrix}.$$

The unnormalized Laplacian matrix is defined as $L_l = D_l - W_l$, where the degree matrix D_l is the diagonal matrix with the degrees $d_j = \sum_{k=1}^{2\mathcal{S}_l} w_{jk}$. The unnormalized Laplacian matrix and its eigenvalues and eigenvectors can be used to effectively embed points in a lower-dimension representation without losing too information about spatial relationships. If neuron clusters can be identified, then the Laplacian L_l is approximately block-diagonal, with each block corresponding a cluster. Assuming there are k clusters in the graph, spectral clustering would then take the first k eigenvectors[1], $\mathcal{U} \in \mathbb{R}^{2\mathcal{S}_l \times k}$, corresponding to the k

smallest eigenvalues, and partition neurons in the eigenspace with the k-means algorithm.

A.3.3 Spectral Clustering Results

We use Net1 and Net2 as an example for showing the results of matching neurons between DNNs. Figure A.5 shows the permuted combined correlation matrix after apply the spectral clustering algorithm for conv1 – conv5. Figure A.6 displays 12 neuron clusters with high between-net similarity measurement in conv1 layer, and Figure A.7 displays the top 8 neuron clusters with highest between-net similarity measurement in conv2 layer. The matching results imply that there exists many-to-many correspondence of the feature maps between two fully trained networks with different random initializations, and the number of neurons learning the same feature can be different between networks. For example, the four units of $\{89, 90, 134, 226\}$ in Net1 and three units of $\{2, 39, 83\}$ in Net2 are learning the features about green objects.

We also computed and visualized the matching neurons in other layers as well. The results of conv1 are shown in Figure A.6. As noted earlier, the conv1 layer tends to learn more general features like Gabor filters (edge detectors) and blobs of color. Our approach finds many matching Gabor filters (e.g., clusters #5, #6, #10, #11 and #12), and also some matching color blobs (e.g., clusters #1, #3 and #4).

[1] In practice, when the number of clusters is unknown, the best value of k to choose is where where the eigenvalue shows a relatively abrupt change.

A.3.4 Hierarchical Spectral Clustering Results

Due to the stochastic effects of randomly initializing centroids in k-means clustering, some of the initial clusters contains more neurons than others. To get more fine-grained cluster structure, we recurrently apply k-means clustering on any clusters with size $> 2\alpha \cdot \mathcal{S}_l$, where α is a tunable parameter for adjusting the maximum size of the leaf clusters. Figure A.8 shows the partial hierarchical structure of neuron matchings in the conv2 layer. The cluster at the root of the tree is a first-level cluster that contains many similar units from both DNNs. Here we adopt $\alpha = 0.025$ for the conv2 layer, resulting in a hierarchical neuron cluster tree structure with leaf clusters containing less than 6 neurons from each network. The bold box of each subcluster contains neurons from Net1 and the remaining neurons are from Net2. For example, in subcluster #3, which shows conv2 features, units $\{62, 137, 148\}$ from Net1 learned similar features as units $\{33, 64, 230\}$ from Net2, namely, red and magenta objects.

A.3.5 Metrics for Neuron Clusters

Here we introduce two metrics for quantifying the similarity among neurons grouped together after applying the clustering algorithm above.

$$\begin{aligned} \text{Between-net similarity:} \quad & \text{Sim}_{X_l \to Y_l} &&= (\sum_{p=1}^{\mathcal{S}_l} \sum_{q=1}^{\mathcal{S}_l} \text{corr}(X_l, Y_l)_{pq}) / \mathcal{S}_l^2 \\ \text{Within-net similarity:} \quad & \text{Sim}_{X_l, Y_l} &&= (\text{Sim}_{X_l \to X_l} + \text{Sim}_{Y_l \to Y_l})/2 \end{aligned}$$

We further performed experiments in quantifying the similarity among neurons that are clustered together. Figure **??** shows the between-net and within-net similarity measurement for conv1 – conv5. The value of k for initial clustering

is set to be 40 for conv1 layer and 100 for all the other layers. In our experiments, the number of final clusters obtained after further hierarchical branching is {43, 113, 130, 155, 131}. The tail in those curves with value 0 is due to the non-existence of between-net similarity for the clusters containing neurons from only one of the two DNNs. To better capture the distribution of non-zero similarity values, we leave out the tail after 100 in the plot for conv3 - conv5 layers.

A.4 Comparing Average Neural Activations within and between Networks

The first layer of networks trained on natural images (here the conv1 layer) tends to learn channels matching patterns similar to Gabor filters (oriented edge filters) and blobs of color. As shown in Figures A.9, A.10, and A.11, there are certain systematic biases in the relative magnitudes of the activations of the different channels of the first layer. Responses of the low frequency filters have much higher magnitude than that of the high frequency filters. This phenomenon is likely a consequence of the $1/f$ power spectrum of natural images in which, on average, low spatial frequencies tend to contain higher energy (because they are more common) than high spatial frequencies.

In Figure A.9 we show the mean activations for each unit of four networks, plotted in sorted order from highest to lowest. First and most saliently, we see a pattern of widely varying mean activation values across units, with a gap between the most active and least active units of one or two orders of magnitude (depending on the layer). Second, we observe a rough overall correspondence

in the spectrum of activations between the networks. However, the correspondence is not perfect: although much of the spectrum matches well, the most active filters converged to solutions of somewhat different magnitudes. For example, the average activation value of the filter on conv2 with the highest average activation varies between 49 to 120 over the four networks; the range for conv1 was 98 to 130.[2] This effect is more interesting considering that all filters were learned with constant weight decay, which pushes all individual filter weights and biases (and thus subsequent activations) toward zero with the same force.

Figure A.10 shows the conv1 units with the highest and lowest activations for each of the four networks. As mentioned earlier (and as expected), filters for lower spatial frequencies have higher average activation, and vice versa. What is surprising is the relative lack of ordering between the four networks. For example, the top two most active filters in Net1 respond to constant color regions of black or light blue, whereas none of the top eight filters in Net2 or Net3 respond to such patterns. One might have thought that whatever influence from the dataset caused the largest filters to be black and blue in the first network would have caused similar constant color patches to dominate the other networks, but we did not observe such consistency. Similar differences exist when observing the learned edge filters: in Net1 and Net4 the most active edge filter is horizontal; in Net2 and Net3 it is vertical. The right half of Figure A.10 depicts the least active filters. The same lack of alignment arises, but here the activation values are more tightly packed, so the exact ordering is less meaningful. Figure A.11 shows the even more widely varying activations from conv2.

[2]Recall that the units use rectified linear activation functions, so the activation magnitude is unbounded. The max activation over all channels and all spatial positions of the first layer is often over 2000.

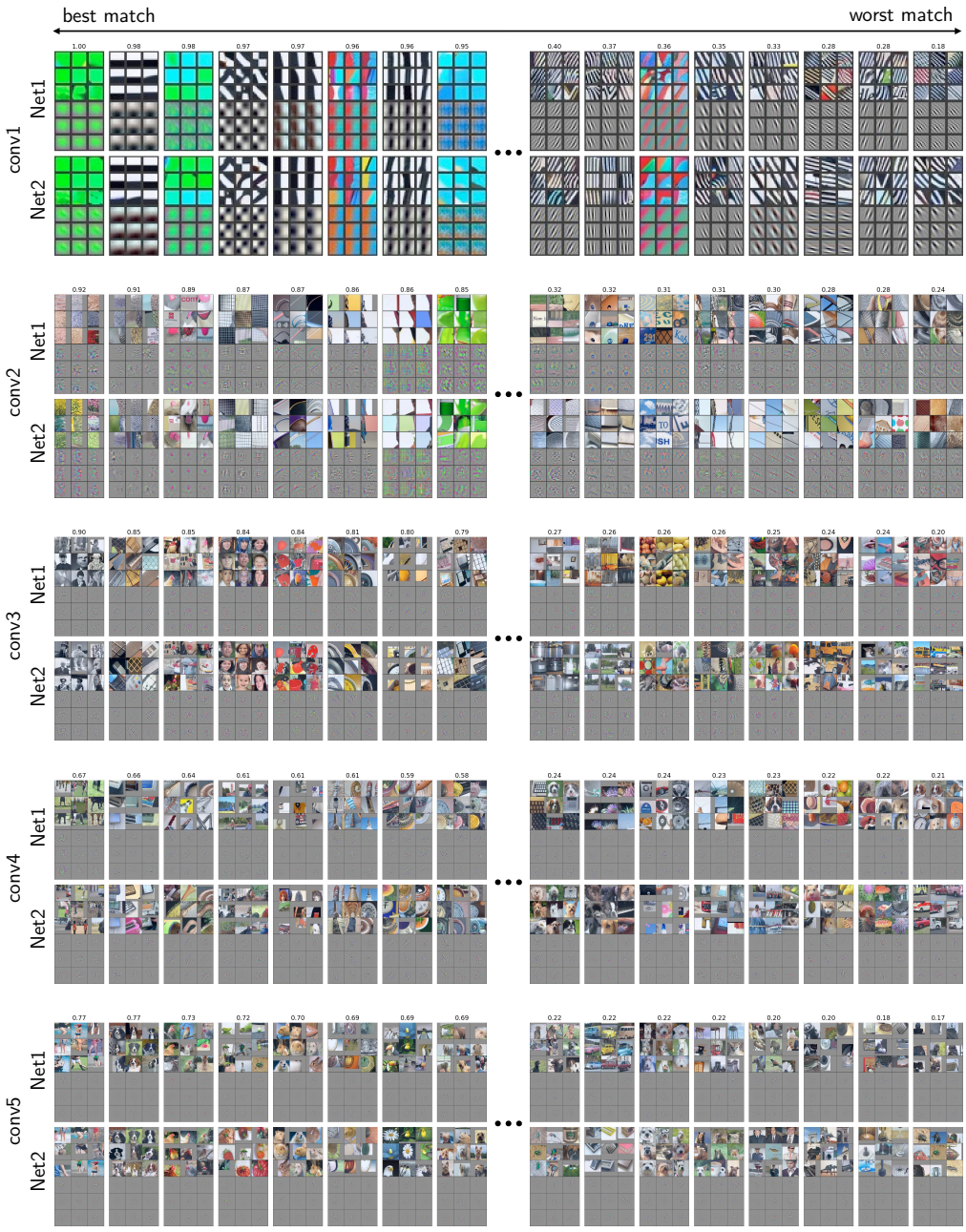


Figure A.2: With assignments chosen by semi-matching, the eight best (highest correlation, left) and eight worst (lowest correlation, right) matched features between Net1 and Net2 for the conv1 through conv5 layers. To visualize the functionality each unit, we plot the nine image patches (in a three by three block) from the validation set that causes the highest activation for that unit and directly beneath that block show the "deconv" visualization of each of the nine images. Best view with siginicant zoom in.

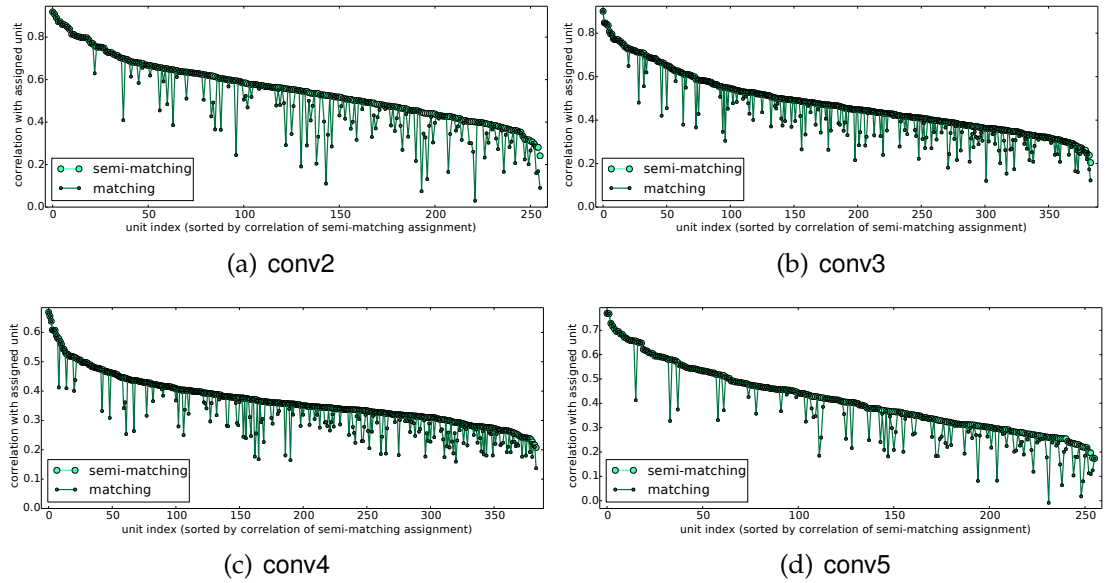


Figure A.3: Correlations between units in conv2 - conv5 layers of Net1 and their paired units in Net2, where pairings are made via semi-matching (large light green circles) or matching (small dark green dots).

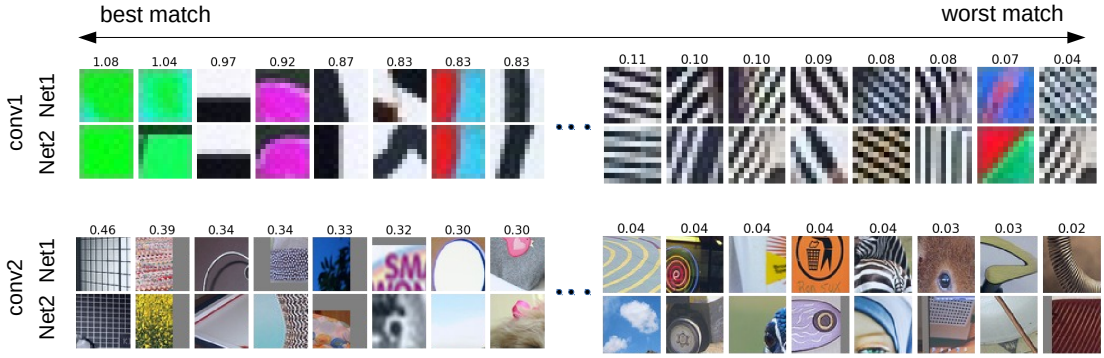
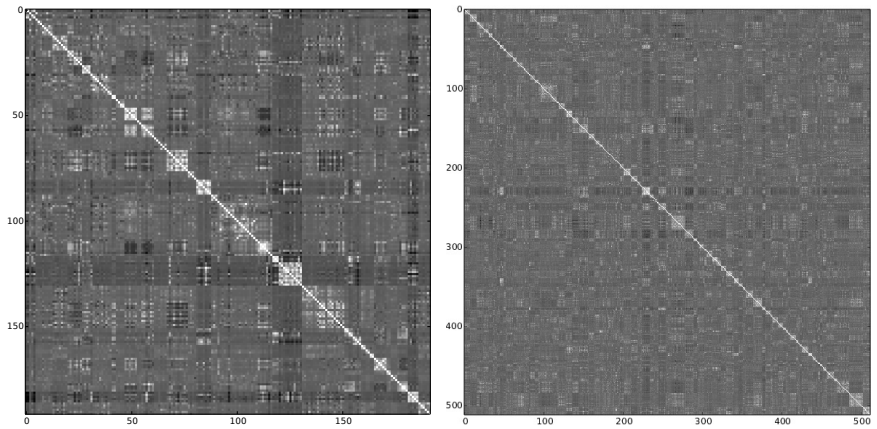
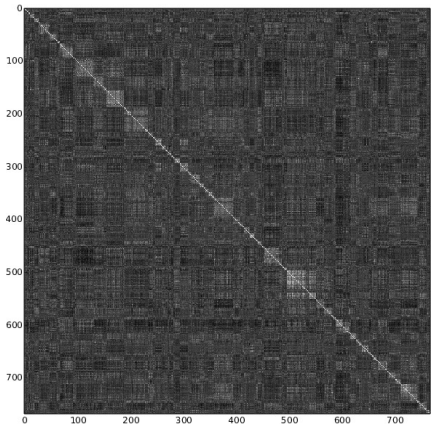


Figure A.4: The eight best (highest mutual information, left) and eight worst (lowest mutual information, right) features in the semi-matching between Net1 and Net2 for the conv1 and conv2 layers.

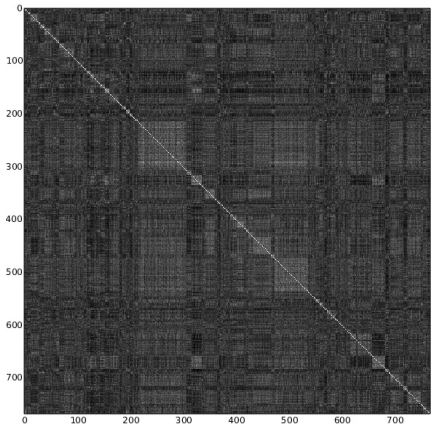


(a) conv1

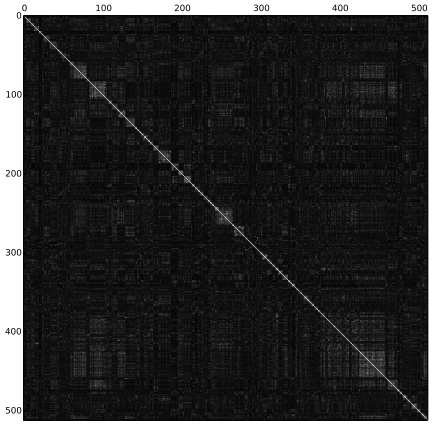
(b) conv2



(c) conv3



(d) conv4



(e) conv5

Figure A.5: The permuted combined correlation matrix after apply spectral clustering method (conv1 – conv5). The diagonal block structure represents the groups of neurons that are clustered together. The value of k adopted for these five layers are: {40,100,100,100,100}, which is consistent with the parameter setting for other experiments in this paper.

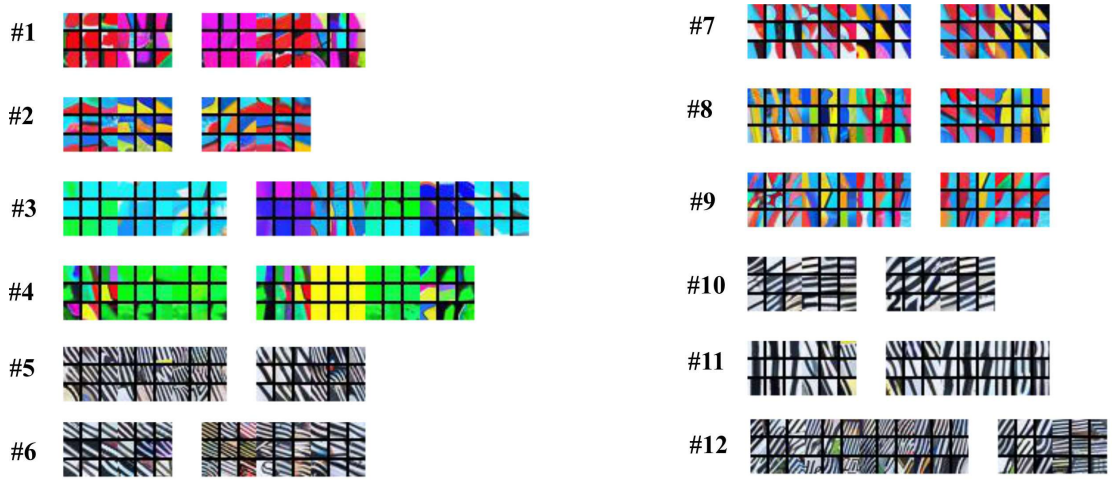


Figure A.6: The neuron matchings between two DNNs (Net1 and Net2) in conv1 layer. Here we display the 12 neuron clusters with relatively high between-net similarity measurement. Each labeled half-row corresponds to one cluster, where the filter visualizations for neurons from Net1 and Net2 are separated by white space slot. The matching results imply that there exists many-to-many correspondence of the feature maps between two fully trained networks with different random initializations. For instance, in cluster #6, neurons from Net1 and Net2 are both learning 135° diagonal edges; and neurons in cluster #10 and #12 are learning 45° diagonal edges.

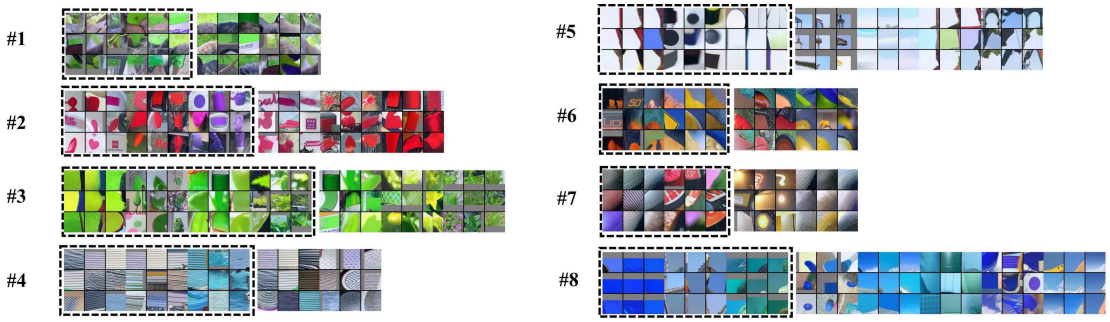


Figure A.7: The neuron matchings between two DNNs: Net1 and Net2. Each of the 3×3 block displays the top 9 image patches that cause the highest activations to each neuron. Each labeled half-row corresponds to one cluster, where the filter visualizations with dashed boxes represent neurons from Net1 and those without are from Net2. For example, there are 7 neurons learning similar features in cluster #3, where the left four neurons are in Net1 and the right three are from Net2. Best viewed in electronic form with zoom.

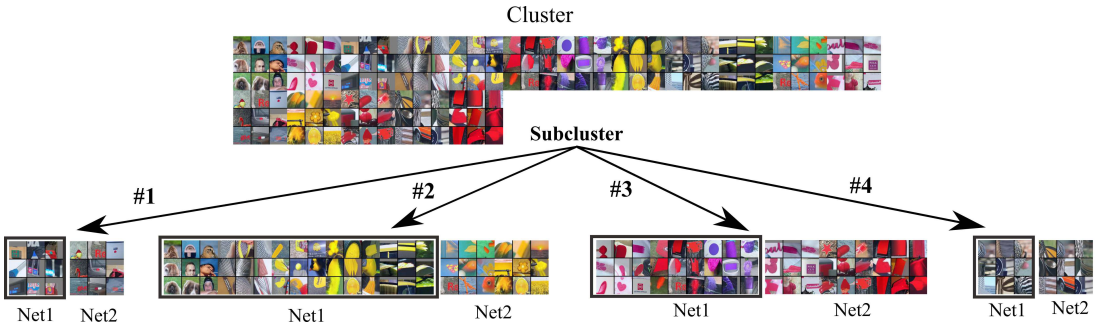


Figure A.8: The hierarchical structure of neuron matchings between two DNNs: Net1 and Net2 (conv2 layer). The initial clusters are obtained using spectral clustering with the number of clusters $k = 100$ and threshold $\tau = 0.2$.

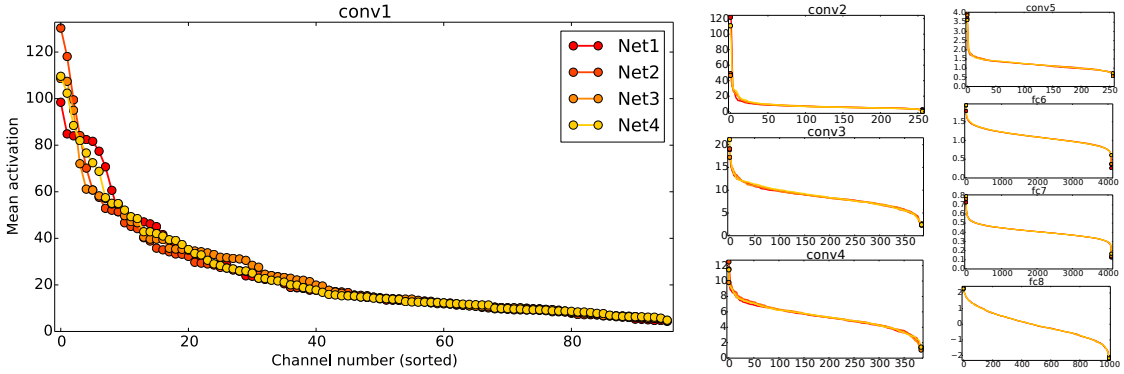


Figure A.9: The average activation values of each unit on all layers of Net1 – Net4. A couple salient effects are observable. First, in a given network, average activations vary widely within each layer. While most activations fall within a relatively narrow band (the middle of each plot), a rare few highly active units have one or two orders of magnitude higher average output than the least active. Second, the overall distribution of activation values is similar across networks. However, also note that the max single activation does vary across networks in some cases, e.g. on the conv2 layer by a factor of two between networks. For clarity, on layers other than conv1 circle markers are shown only at the line endpoints.

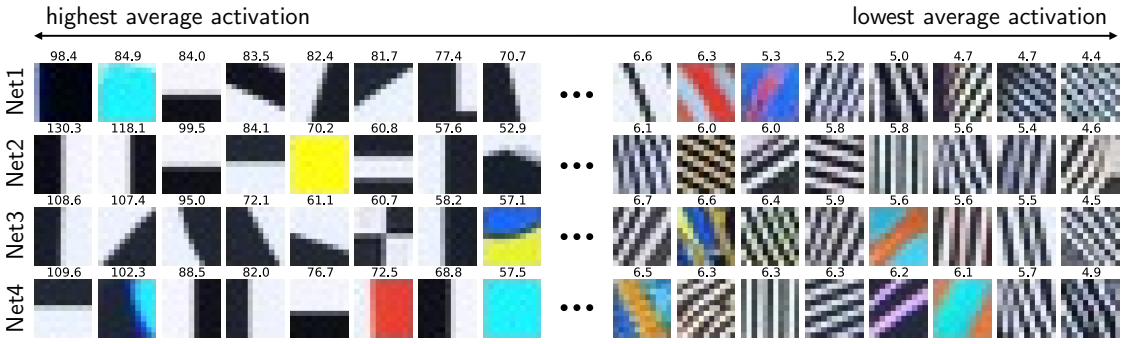


Figure A.10: The most active (left) to least active (right) conv1 filters from Net1 – Net4, with average activation values printed above each filter. The most active filters generally respond to low spatial frequencies, and the least active filtered to high spatial frequencies, but the lack of alignment is interesting (see text).

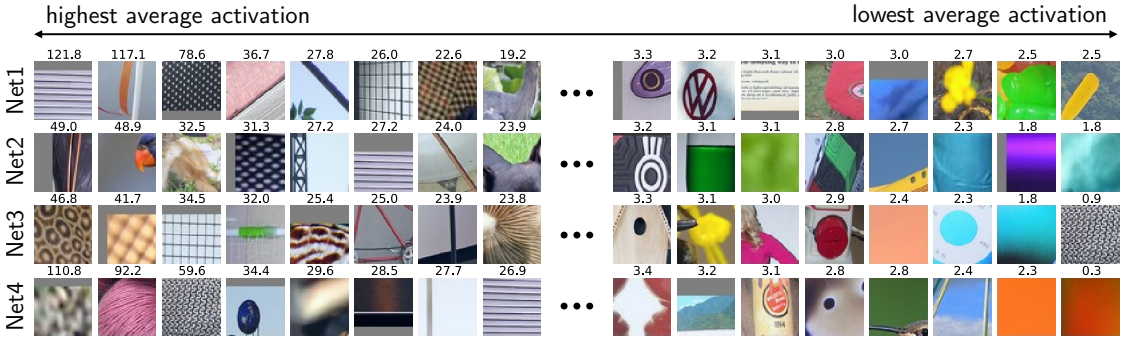


Figure A.11: Most active (left) to least active (right) conv2 filters as in Figure A.10. Compared to the conv1 filters, here the separation and misalignment between the top filters is even larger. For example, the top unit responding to horizontal lines in Net1 has average activation of 121.8, whereas similar units in Net2 and Net4 average 27.2 and 26.9, respectively. The unit does not appear in the top eight units of Net3 at all. The least active units seem to respond to rare specific concepts.

APPENDIX B

APPENDIX FOR CHAPTER 6

B.1 Single model and Snapshot Ensemble performance over time

We present more details about experimental results of Snapshot Ensembles in Figure B.1-B.3. The blue curve shows how the test error of single model snapshot using cyclical cosine learning rate. The green curve shows the test error by ensembling model snapshots over time. As a reference, the red dashed line in each panel represents the test error of single model trained for 300 epochs using standard learning rate schedule. Without Snapshot Ensembles, we find that in about half of the cases, the test error of final model using cyclical learning rate (shown on the right most point in the blue curve) is no better than using standard learning rate scheduling.

One can observe that under all settings except ResNet-110 on CIFAR-10, Snapshot Ensembles (shown on the right most point of the green curves) outperform the single model baselines. In many cases, ensemble of just 2 or 3 model snapshots is able to match the performance of the single model trained with standard learning rate. Not surprisingly, ensemble of model snapshots consistently outperform any of its members, yielding a smooth curve of test error over time.

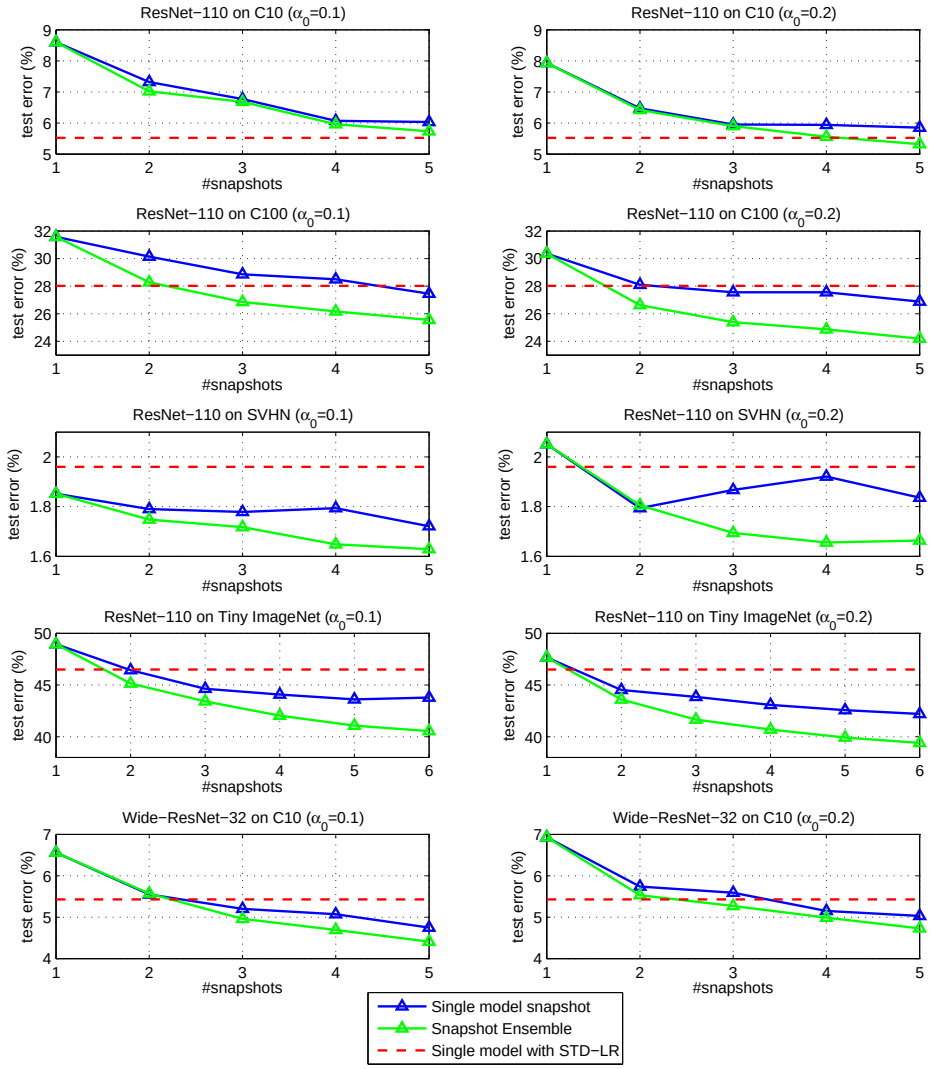


Figure B.1: Single model and Snapshot Ensemble performance over time.

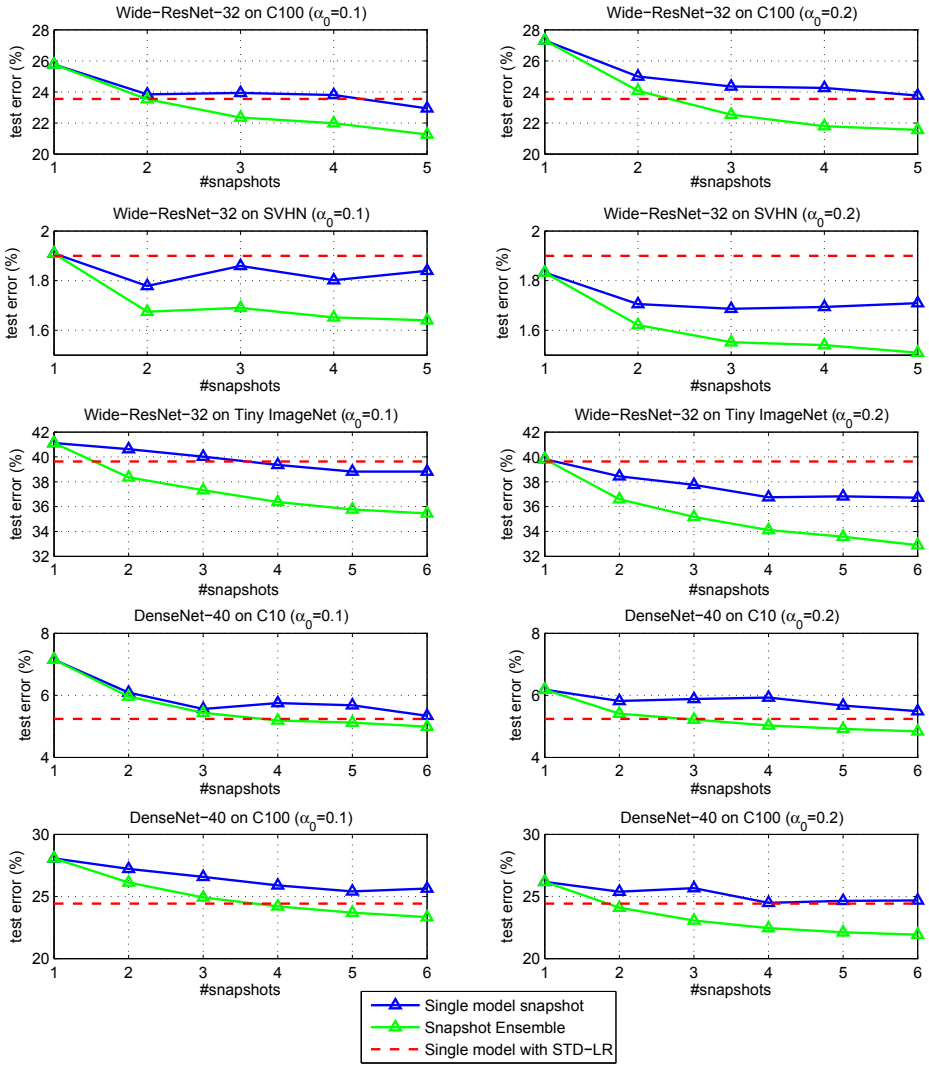


Figure B.2: Single model and Snapshot Ensemble performance over time.

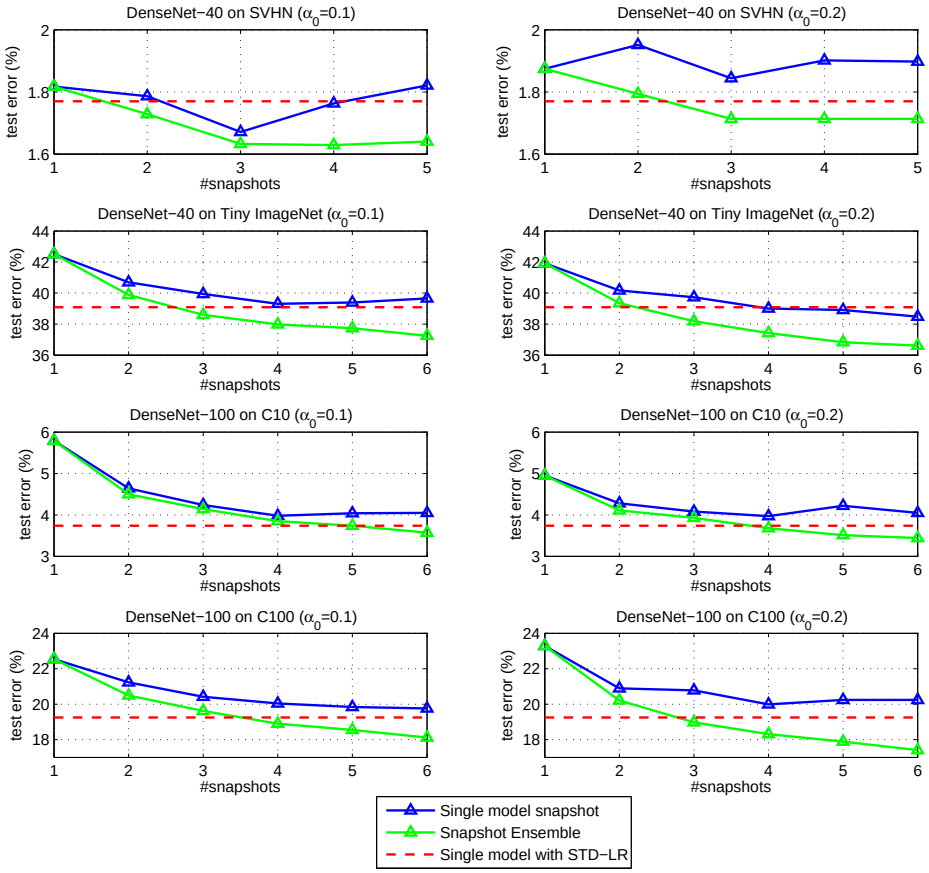


Figure B.3: Single model and Snapshot Ensemble performance over time.

BIBLIOGRAPHY

[1]

[2] WeChat group chat features. www.wechat.com/en/features.html#group.

[3] WeChat wiki. en.wikipedia.org/wiki/WeChat.

[4] Youtube: Get legitimate subscribers. https://support.google.com/youtube/answer/6051134.

[5] Youtube: Spam, deceptive practices, and scams. https://support.google.com/youtube/answer/2801973.

[6] Bruno Abrahao, Sucheta Soundarajan, John Hopcroft, and Robert Kleinberg. A separability framework for analyzing community structure. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(1):5, 2014.

[7] Eytan Adar, Li Zhang, Lada A Adamic, and Rajan M Lukose. Implicit structure and the dynamics of blogspace. In *Workshop on the weblogging ecosystem*, pages 16989–16995, 2004.

[8] Yong-Yeol Ahn, James P Bagrow, and Sune Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466(7307):761–764, 2010.

[9] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486. IEEE, 2006.

[10] Reid Andersen and Kevin J Lang. Communities from seed sets. In *WWW*, pages 223–232. ACM, 2006.

[11] Ashton Anderson, Daniel Huttenlocher, Jon Kleinberg, Jure Leskovec, and Mitul Tiwari. Global diffusion via cascading invitations: Structure, growth, and homophily. In *WWW*, pages 66–76, 2015.

[12] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *The Journal of Machine Learning Research*, 6:1817–1853, 2005.

[13] Rie Kubota Ando and Tong Zhang. A high-performance semi-supervised learning method for text chunking. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 1–9. Association for Computational Linguistics, 2005.

[14] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2425–2433, 2015.

[15] Sinan Aral, Lev Muchnik, and Arun Sundararajan. Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks. *Proceedings of the National Academy of Sciences*, 106(51):21544–21549, 2009.

[16] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. In *ICML*, pages 584–592, 2014.

[17] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, pages 44–54. ACM, 2006.

[18] Eytan Bakshy, Itamar Rosenn, Cameron Marlow, and Lada Adamic. The role of social networks in information diffusion. In *WWW*, pages 519–528, 2012.

[19] Luca Becchetti, Carlos Castillo, Debora Donato, Ricardo Baeza-Yates, and Stefano Leonardi. Link analysis for web spam detection. *ACM Transactions on the Web (TWEB)*, 2(1):2, 2008.

[20] Fabricio Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgilio Almeida. Detecting spammers on twitter. In *Collaboration, electronic messaging, anti-abuse and spam conference (CEAS)*, volume 6, page 12, 2010.

[21] Fabrício Benevenuto, Tiago Rodrigues, Virgílio Almeida, Jussara Almeida, and Marcos Gonçalves. Detecting spammers and content promoters in online video social networks. In *SIGIR*, pages 620–627. ACM, 2009.

[22] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[23] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*, pages 119–130. ACM, 2013.

[24] John Blitzer. *Domain adaptation of natural language processing systems*. PhD thesis, University of Pennsylvania, 2007.

[25] John Blitzer, Sham Kakade, and Dean P Foster. Domain adaptation with coupled subspaces. In *International Conference on Artificial Intelligence and Statistics*, pages 173–181, 2011.

[26] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 120–128. Association for Computational Linguistics, 2006.

[27] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT*. 2010.

[28] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[29] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[30] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Knowledge discovery and data mining*, 2006.

[31] Cody Buntain and Jennifer Golbeck. Identifying social roles in reddit using network structure. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 615–620. ACM, 2014.

[32] Brian S Butler. Membership size, communication activity, and sustainability: A resource-based model of online social structures. *Information systems research*, 12(4):346–362, 2001.

[33] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. Uncovering large groups of active malicious accounts in online social networks. In *SIGSAC CCS*, pages 477–488. ACM, 2014.

[34] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *International Conference on Machine Learning*, 2004.

[35] Carlos Castillo, Debora Donato, Aristides Gionis, Vanessa Murdock, and Fabrizio Silvestri. Know your neighbors: Web spam detection using the web topology. In *SIGIR*, pages 423–430. ACM, 2007.

[36] Meeyoung Cha, Alan Mislove, Ben Adams, and Krishna P Gummadi. Characterizing social cascades in flickr. In *Proceedings of the first workshop on Online social networks*, pages 13–18, 2008.

[37] Justin Cheng, Lada Adamic, P Alex Dow, Jon Michael Kleinberg, and Jure Leskovec. Can cascades be predicted? In *WWW*, pages 925–936. ACM, 2014.

[38] Paul-Alexandru Chirita, Jörg Diederich, and Wolfgang Nejdl. Mailrank: using ranking for spam detection. In *CIKM*, pages 373–380. ACM, 2005.

[39] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[40] Karen Church and Rodrigo de Oliveira. What's up with whatsapp?: comparing mobile instant messaging behaviors with traditional sms. In *Mobile-CHI*, pages 352–361. ACM, 2013.

[41] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[42] Michele Coscia, Giulio Rossetti, Fosca Giannotti, and Dino Pedreschi. Demon: a local-first discovery method for overlapping communities. In *KDD*, pages 615–623. ACM, 2012.

[43] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle

point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[44] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

[45] Emiliano De Cristofaro, Arik Friedman, Guillaume Jourjon, Mohamed Ali Kaafar, and M Zubair Shafiq. Paying for likes?: Understanding facebook like fraud using honeypots. In *IMC*, pages 129–136. ACM, 2014.

[46] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[47] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*, 2009.

[48] Harris Drucker, Corinna Cortes, Lawrence D Jackel, Yann LeCun, and Vladimir Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301, 1994.

[49] Nicolas Ducheneaut, Nicholas Yee, Eric Nickell, and Robert J Moore. The life and death of online gaming communities: a look at guilds in world of warcraft. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 839–848, 2007.

[50] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[51] David Eigen, Jason Rolfe, Rob Fergus, and Yann LeCun. Understanding deep architectures using a recursive convolutional network. *arXiv preprint arXiv:1312.1847*, 2013.

[52] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *International Conference on artificial intelligence and statistics*, pages 153–160, 2009.

[53] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[54] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[55] Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh K Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C Platt, et al. From captions to visual concepts and back. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1473–1482, 2015.

[56] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[57] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model.

In *Advances in Neural Information Processing Systems*, pages 2121–2129, 2013.

[58] Hongyu Gao, Jun Hu, Christo Wilson, Zhichun Li, Yan Chen, and Ben Y Zhao. Detecting and characterizing social spam campaigns. In *IMC*, pages 35–47. ACM, 2010.

[59] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[60] David F Gleich and Michael W Mahoney. Using local spectral methods to robustify graph-based learning algorithms. In *SIGKDD*, pages 359–368. ACM, 2015.

[61] Sharad Goel, Ashton Anderson, Jake Hofman, and Duncan Watts. The structural virality of online diffusion. *Preprint*, 22:26, 2013.

[62] Manuel Gomez Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1019–1028. ACM, 2010.

[63] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[64] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

[65] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville,

and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[66] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[67] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12:993–1001, 1990.

[68] Mohamad H Hassoun. *Fundamentals of artificial neural networks*. MIT press, 1995.

[69] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[70] Kun He, Yiwei Sun, David Bindel, John Hopcroft, and Yixuan Li. Detecting overlapping communities from local spectral subspaces. In *ICDM*. ACM, 2015.

[71] Paul Heymann, Georgia Koutrika, and Hector Garcia-Molina. Fighting spam on social web sites: A survey of approaches and future challenges. *Internet Computing, IEEE*, 11(6):36–45, 2007.

[72] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[73] Paul W Holland and Samuel Leinhardt. Transitivity in structural models of small groups. *Comparative Group Studies*, 1971.

[74] John Hopcroft, Omar Khan, Brian Kulis, and Bart Selman. Natural communities in large linked networks. In *Proceedings of the ninth ACM*

SIGKDD international conference on Knowledge discovery and data mining, pages 541–546. ACM, 2003.

[75] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[76] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[77] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth. *arXiv preprint arXiv:1603.09382*, 2016.

[78] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[79] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[80] Sbastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

[81] Shuiwang Ji, Lei Tang, Shipeng Yu, and Jieping Ye. A shared-subspace learning framework for multi-label classification. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(2):8, 2010.

[82] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Con-

volutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[83] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Inferring strange behavior from connectivity pattern in social networks. In *Advances in Knowledge Discovery and Data Mining*, pages 126–138. Springer, 2014.

[84] Di Jin, Bo Yang, Carlos Baquero, Dayou Liu, Dongxiao He, and Jie Liu. A markov random walk under constraint for discovering overlapping communities in complex networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(05):P05031, 2011.

[85] Thorsten Joachims. *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

[86] Sanjay Ram Kairam, Dan J Wang, and Jure Leskovec. The life and death of online groups: Predicting group growth and longevity. In *WSDM*, pages 673–682. ACM, 2012.

[87] Sham M Kakade and Dean P Foster. Multi-view regression via canonical correlation analysis. In *Learning theory*, pages 82–96. Springer, 2007.

[88] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[89] Andrej Karpathy, Armand Joulin, and Fei Fei F Li. Deep fragment embeddings for bidirectional image sentence mapping. In *Advances in neural information processing systems*, pages 1889–1897, 2014.

[90] Kenji Kawaguchi. Deep learning without poor local minima. *arXiv preprint arXiv:1605.07110*, 2016.

[91] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[92] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[93] Kyle Kloster and David F Gleich. Heat kernel based community detection. In *KDD*. ACM, 2014.

[94] Isabel Kloumann, Lada Adamic, Jon Kleinberg, and Shaomei Wu. The lifecycles of apps in a social ecosystem. In *WWW*, pages 581–591, 2015.

[95] Isabel M. Kloumann and Jon M. Kleinberg. Community membership identification from small seed sets. In *KDD*. ACM, 2014.

[96] Pranam Kolari, Akshay Java, Tim Finin, Tim Oates, and Anupam Joshi. Detecting spam blogs: A machine learning approach. In *AAAI*, volume 21, page 1351, 2006.

[97] Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81, 2009.

[98] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[99] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[100] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[101] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[102] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.

[103] Anders Krogh, Jesper Vedelsby, et al. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, volume 7, 1995.

[104] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron Courville, et al. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

[105] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *ICML*, volume 97, pages 179–186, 1997.

[106] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *WWW*, pages 591–600. ACM, 2010.

[107] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016.

[108] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, 2008.

[109] Andrea Lancichinetti, Filippo Radicchi, José J Ramasco, and Santo Fortunato. Finding statistically significant communities in networks. *PloS one*, 6(4):e18961, 2011.

[110] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[111] Eugene L Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 2001.

[112] Quoc V Le, Alexandre Karpenko, Jiquan Ngiam, and Andrew Y Ng. Ica with reconstruction cost for efficient overcomplete feature learning. In *Advances in Neural Information Processing Systems*, pages 1017–1025, 2011.

[113] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[114] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. 2015.

[115] Conrad Lee, Fergal Reid, Aaron McDaid, and Neil Hurley. Detecting highly overlapping community structure by greedy clique expansion. *arXiv preprint arXiv:1002.1827*, 2010.

[116] Kyumin Lee, James Caverlee, Krishna Y Kamath, and Zhiyuan Cheng. Detecting collective attention spam. In *WebQuality*, pages 48–55. ACM, 2012.

[117] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 991–999, 2015.

[118] Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5, 2007.

[119] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, pages 695–704. ACM, 2008.

[120] Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie S Glance, and Matthew Hurst. Patterns of cascading behavior in large blog graphs. In *SDM*, volume 7, pages 551–556. SIAM, 2007.

[121] Yixuan Li, Kun He, David Bindel, and John E Hopcroft. Uncovering the small community structure in large networks: A local spectral approach. In *WWW*, pages 658–668. ACM, 2015.

[122] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent learning: Do different neural networks learn the same representations? *arXiv preprint arXiv:1511.07543*, 2015.

[123] Ee-Peng Lim, Viet-An Nguyen, Nitin Jindal, Bing Liu, and Hady Wirawan Lauw. Detecting product review spammers using rating behaviors. In *CIKM*, pages 939–948. ACM, 2010.

[124] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[125] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[126] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. *arXiv preprint arXiv:1412.0035*, 2014.

[127] Michael W Mahoney, Lorenzo Orecchia, and Nisheeth K Vishnoi. A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally. *The Journal of Machine Learning Research*, 13(1):2339–2365, 2012.

[128] Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. Ask your neurons: A neural-based approach to answering questions about images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1–9, 2015.

[129] Bhaskar Mehta, Saurabh Nangia, Manish Gupta, and Wolfgang Nejdl. Detecting image spam using visual features and near duplicate detection. In *WWW*, pages 497–506. ACM, 2008.

[130] Marina Meila and Jianbo Shi. A random walks view of spectral segmentation. 2001.

[131] Mohammad Moghimi, Mohammad Saberian, Jian Yang, Li-Jia Li, Nuno Vasconcelos, and Serge Belongie. Boosted convolutional neural networks.

[132] Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random structures & algorithms*, 6(2-3):161–180, 1995.

[133] Grégoire Montavon, Mikio L Braun, and Klaus-Robert Müller. Kernel analysis of deep networks. *The Journal of Machine Learning Research*, 12:2563–2581, 2011.

[134] Arjun Mukherjee, Bing Liu, and Natalie Glance. Spotting fake reviewer groups in consumer reviews. In *WWW*, pages 191–200. ACM, 2012.

[135] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning, 2011. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[136] Mark EJ Newman. Detecting community structure in networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):321–330, 2004.

[137] Behnam Neyshabur and Rina Panigrahy. Sparse matrix factorization. *arXiv preprint arXiv:1311.3315*, 2013.

[138] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[139] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

[140] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *arXiv preprint arXiv:1412.1897*, 2014.

[141] Alexandros Ntoulas, Marc Najork, Mark Manasse, and Dennis Fetterly. Detecting spam web pages through content analysis. In *WWW*, pages 83–92. ACM, 2006.

[142] Guillaume Obozinski, Ben Taskar, and Michael I Jordan. Joint covariate selection and joint subspace selection for multiple classification problems. *Statistics and Computing*, 20(2):231–252, 2010.

[143] Derek O'Callaghan, Martin Harrigan, Joe Carthy, and Pádraig Cunningham. Network analysis of recurring youtube spam campaigns. In *ICWSM*, 2012.

[144] Gergely Palla, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. *Nature*, 446(7136):664–667, 2007.

[145] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *WWW*, pages 201–210. ACM, 2007.

[146] Namsu Park, Kerk F Kee, and Sebastián Valenzuela. Being immersed in social networking environment: Facebook groups, uses and gratifications, and social outcomes. *CyberPsychology & Behavior*, 12(6):729–733, 2009.

[147] Arnab Paul and Suresh Venkatasubramanian. Why does deep learning work?-a perspective from group theory. *arXiv preprint arXiv:1412.6621*, 2014.

[148] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[149] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. In *Computer and Information Sciences-ISCIS 2005*, pages 284–293. Springer, 2005.

[150] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[151] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

[152] Bruno Ribeiro. Modeling and predicting the growth and death of membership-based websites. In *WWW*, pages 653–664. ACM, 2014.

[153] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[154] Daniel M Romero, Brendan Meeder, and Jon Kleinberg. Differences in the mechanics of information diffusion across topics: idioms, political hashtags, and complex contagion on twitter. In *WWW*, pages 695–704. ACM, 2011.

[155] Martin Rosvall and Carl T Bergstrom. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PloS one*, 6(4):e18209, 2011.

[156] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[157] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[158] Rico Sennrich, Barry Haddow, and Alexandra Birch. Edinburgh neural machine translation systems for wmt 16. *arXiv preprint arXiv:1606.02891*, 2016.

[159] Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. In *International Conference on Pattern Recognition*, 2012.

[160] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[161] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[162] Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. *arXiv preprint arXiv:1605.06465*, 2016.

[163] Leslie N. Smith. No more pesky learning rate guessing games. *CoRR*, abs/1506.01186, 2016.

[164] Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2:207–218, 2014.

[165] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*, pages 81–90. ACM, 2004.

[166] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[167] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[168] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[169] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *ACSAC*, pages 1–9. ACM, 2010.

[170] Gianluca Stringhini, Pierre Mourlanne, Gregoire Jacob, Manuel Egele, Christopher Kruegel, and Giovanni Vigna. Evilcohort: detecting communities of malicious accounts on online services. In *USENIX Security Symposium*, 2015.

[171] Eric Sun, Itamar Rosenn, Cameron Marlow, and Thomas M Lento. Gesundheit! modeling contagion through facebook news feed. In *ICWSM*, 2009.

[172] Yizhou Sun, Jie Tang, Jiawei Han, Cheng Chen, and Manish Gupta. Co-evolution of multi-typed objects in dynamic star networks. *IEEE TKDE*, 26(12):2942–2955, 2014.

[173] A Swann and N Allinson. Fast committee learning: Preliminary results. *Electronics Letters*, 34(14):1408–1410, 1998.

[174] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[175] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[176] Johan Ugander, Lars Backstrom, Cameron Marlow, and Jon Kleinberg. Structural diversity in social contagion. *Proceedings of the National Academy of Sciences*, 109(16):5962–5966, 2012.

[177] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[178] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, 2013.

[179] Alex Hai Wang. Don't follow me: Spam detection in twitter. In *SECRYPT*, pages 1–10. IEEE, 2010.

[180] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y Zhao. You are how you click: Clickstream analysis for sybil detection. In *Usenix Security*, pages 241–256, 2013.

[181] Yi-Min Wang, Ming Ma, Yuan Niu, and Hao Chen. Spam double-funnel: Connecting web spammers with advertisers. In *WWW*, pages 291–300. ACM, 2007.

[182] Duncan J Watts. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences*, 99(9):5766–5771, 2002.

[183] Lilian Weng, Filippo Menczer, and Yong-Yeol Ahn. Virality prediction and community structure in social networks. *Scientific reports*, 3, 2013.

[184] Joyce Jiyoung Whang, David F Gleich, and Inderjit S Dhillon. Overlapping community detection using seed set expansion. In *CIKM*, pages 2099–2108. ACM, 2013.

[185] Baoning Wu, Vinay Goel, and Brian D Davison. Topical trustrank: Using topicality to combat web spam. In *WWW*, pages 63–72. ACM, 2006.

[186] Ching-Tung Wu, Kwang-Ting Cheng, Qiang Zhu, and Yi-Leh Wu. Using visual features for anti-spam filtering. In *ICIP*, volume 3, pages III–509. IEEE, 2005.

[187] Jierui Xie, Stephen Kelley, and Boleslaw K Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Computing Surveys (CSUR)*, 45(4):43, 2013.

[188] Jingjing Xie, Bing Xu, and Zhang Chuang. Horizontal and vertical ensemble with deep representation for classification. *arXiv preprint arXiv:1306.2759*, 2013.

[189] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

[190] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *ICDM*, pages 10–13, December 2012.

[191] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

[192] Jiang Yang, Xiao Wei, Mark S Ackerman, and Lada A Adamic. Activity lifespan: An analysis of user survival patterns in online knowledge sharing communities. In *ICWSM*, 2010.

[193] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y Zhao, and Yafei Dai. Uncovering social network sybils in the wild. *TKDD*, 8(1):2, 2014.

[194] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[195] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[196] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[197] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer vision–ECCV 2014*, pages 818–833. Springer, 2014.

[198] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.

[199] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1):239–263, 2002.

[200] Vinko Zlatić, Andrea Gabrielli, and Guido Caldarelli. Topologically biased random walk and community finding in networks. *Physical Review E*, 82(6):066109, 2010.